Query Evaluation under Differential Privacy

Wei Dong
Hong Kong University of Science and Technology
wdongac@cse.ust.hk

Ke Yi
Hong Kong University of Science and Technology
yike@cse.ust.hk

ABSTRACT

Differential privacy has garnered significant attention in recent years due to its potential in offering robust privacy protection for individual data during analysis. With the increasing volume of sensitive information being collected by organizations and analyzed through SQL queries, the development of a general-purpose query engine that is capable of supporting a broad range of queries while maintaining differential privacy has become the holy grail in privacy-preserving query release. Towards this goal, this article surveys recent advances in query evaluation under differential privacy.

1. INTRODUCTION

Suppose a data analyst is interested in the total number of items sold this year where the customer and supplier are from the same nation. S/he would issue the following SQL query (assuming the TPC-H schema):

```
SELECT count(*)
FROM Customer, Orders, Supplier, Lineitem
WHERE Orders.Orderdate > 2023-01-01
  AND Lineitem.SK = Supplier.SK
  AND Supplier.NK = Customer.NK
  AND Customer.CK = Orders.CK
  AND Orders.OK = Lineitem.OK;
```

Such queries are very common in today's data analytical tasks and are a central problem in databases, which have been extensively studied in the literature. Sophisticated query processing algorithms and systems have been and are continually being developed and optimized throughout the years.

In recent years, a new direction for query processing concerns with the problem of how to release query results while respecting the privacy of the individuals who have contributed their data to the database. For example, the query above clearly relies on the data from the customers and suppliers, and it has been shown that, if the results of a certain number of such queries are available, then the data of the customers/suppliers can be reconstructed with enough accuracy [15]. Meanwhile, privacy-protection laws, such as the General data protection regulation (GDPR) [43], have been enacted across the world, making it a legal responsibility that companies and governments must handle personal data carefully.

Among the many privacy definitions, *differential privacy* (DP) [24] has become the de facto standard for privacy-preserving query release. It requires that the presence or absence of any individual's data should not change the distribution of the query result significantly, so that the adversary cannot infer (with a certain level of confidence) whether any individual has contributed to the database or not. More formally, let $\mathcal{I}$ be the space of all database instances, Q a query, and $M_Q : \mathcal{I} \to \mathcal{Y}$ a query-answering algorithm, often called a *mechanism* in the DP literature. The mechanism M_Q is said to satisfy (ε, δ)-DP if

$$\mathsf{Pr}[M_Q(\mathbf{I}) \in Y] \leq e^{\varepsilon} \cdot \mathsf{Pr}[M_Q(\mathbf{I}') \in Y] + \delta \qquad (1)$$

for any subset of outputs $Y \subseteq \mathcal{Y}$ and any pair of *neighboring instances* $\mathbf{I} \sim \mathbf{I}'$ (to be elaborated shortly). Here, ε, δ are the privacy parameters, also called the privacy budget. Typically, ε is a constant ranging from 0.1 to 10, with smaller values corresponding to stronger privacy guarantees. On the other hand, δ should be much smaller than $1/N$ to ensure the privacy of individual tuples, where $N = |\mathbf{I}|$ is the instance size; in particular, the case where $\delta = 0$ is referred to as *pure DP*, which is more desirable. Note that a DP mechanism must be randomized by definition, and some noise has to be injected to the true query result $Q(\mathbf{I})$. Thus, the central problem in DP is to find the optimal trade-off between privacy (i.e., ε, δ) and utility (i.e., how much noise is injected).

1.1 DP Policies in Relational Databases

The unspecified neighboring relationship $\mathbf{I} \sim \mathbf{I}'$ in the definition above depends on the data model and privacy requirement. For a single table (relation), the standard definition is that $\mathbf{I} \sim \mathbf{I}'$ if one contains one more tuple than the other. However, in a database with multiple relations possibly with foreign-key (FK) constraints, the situation is more subtle. Two neighboring relationships have been proposed and extensively studied, resulting in two different DP policies: tuple-DP [33, 41, 37, 42, 28, 19, 20] and user-DP [45, 34, 16, 18]. The nomenclature of these two policies reflects their respective aims: tuple-DP safeguards the tuples, while user-DP preserves the privacy of users, who may possess multiple tuples.

Tuple-DP is a straightforward generalization of the single-table case: neighboring instances differ by the

addition or deletion of a single tuple in any relation, while FK constraints are ignored. In contrast, user-DP employs a more intricate definition of neighboring instances by taking the FK constraints into consideration. First, one or more relations are designated as the *primary private relations*, whose tuples are the "users" whose information we aim to protect, such as `Customer` and/or `Supplier`. Then, any relation that has an FK reference, direct or indirect, to a primary private relation is called a *secondary private relation*. A tuple in a secondary relation that has an FK reference (directly or indirectly) to a user, such as a lineitem in an order placed by a customer, is considered as data belonging to the user. Relations having no FK references to the primary private relations are public. Then $\mathbf{I}$ and $\mathbf{I}'$ are neighbors if one can be obtained from the other by deleting one user from the primary private relation and all his/her data from the secondary private relations.

Note that when all relations are taken as primary private relations and there are no FK constraints, user-DP degenerates into tuple-DP. Thus, user-DP is more general, hence more difficult to achieve and often makes the utility worse, but it offers stronger and more flexible privacy policies. Which policy to adopt depends on what information is considered private and needs protection. For example, in the TPC-H schema, tuple-DP only protects the privacy of tuples in the relations, such as whether a customer has placed a particular order, whether a particular item is in a given order, and whether a supplier provides a certain item. In contrast, user-DP protects all information about each customer/supplier. Note that when applied to the database schema $\{\texttt{Edge}(\texttt{src}, \texttt{dst}), \texttt{Node}(\texttt{ID})\}$, where `src` and `dst` have FK references to `ID`, user-DP degenerates into *node-DP* (by designating `Node` as the primary private relation) [31, 10, 14] while tuple-DP becomes *edge-DP* [39, 10, 47, 30], both of which have been extensively studied in private graph analysis.

1.2 Classification of Queries

A wide range of queries have been studied under both tuple-DP and user-DP. Below we classify them according to the operators used: Selection, Projection, Join, and Aggregation (Count, Sum, and Max/Min). We assume that the aggregation attribute takes values from the non-negative integer domain[1] in this article, as most works do in this area. Note that for queries that return a subset of the tuples from the input, such as plain conjunctive queries, DP is hard to achieve, so they have not been considered in the literature.

SA Queries.

An SA query imposes a selection condition on a single relation, followed by an aggregation. The following is an example:

```
SELECT count(*) FROM Lineitem
WHERE Lineitem.Shipdate > 2023-01-01;
```

[1]The integer domain can be handled by separately processing the query for the non-negative and negative domains.

While SA queries have been extensively studied under tuple-DP, they are not considered under user-DP. To see why, consider the query above in a TPC-H database where `Customer` is designated as the primary private relation. Such a query is said to be *incomplete* under user-DP, as it does not include the primary private relation, which contains information of the users whose privacy we aim to protect. Thus, under user-DP, the query must first be made complete by iteratively adding relations whose PKs are referenced, together with the necessary PK-FK join condition, until the primary private relations are included (if possible). For example, the query above should be augmented to the following query, which becomes an SJA query:

```
SELECT count(*)
FROM Customer, Orders, Lineitem
WHERE Lineitem.Shipdate > 2023-01-01
  AND Customer.CK = Orders.CK
  AND Orders.OK = Lineitem.OK;
```

SJA Queries.

An SJA query uses selections, joins, followed by an aggregation. The query above is one such example. This example has no self-joins. On the other hand, many useful queries, such as pattern counting queries in a graph, extensively use self-joins. Another scenario where self-joins arise implicitly is user-DP with multiple primary private relations, as this case is first reduced to a single primary private relation. For example, when both `Customer` and `Supplier` are primary private relations, a virtual relation $\texttt{User}(\underline{\texttt{ID}})$ is built that contains all the PKs of these two relations while `Customer.CK` and `Supplier.SK` have FK references to `User.ID`. Then `User` is designated as the only primary private relation, while `Customer` and `Supplier` become secondary private relations. Now consider a query that involves both `Customer` and `Supplier`. After making the query complete as described above, the query contains a self-join on `User`, even if it is self-join-free originally. As will be seen later, the presence of self-joins makes the problem significantly more difficult, thus is often treated separately.

SPJA Queries.

Finally, the query may involve a (distinct) projection on certain attributes before the aggregation; the following is an example.

```
SELECT count(DISTINCT Customer.CK)
FROM Customer, Orders, Lineitem
WHERE Lineitem.Shipdate > 2023-01-01
  AND Customer.CK = Orders.CK
  AND Orders.OK = Lineitem.OK;
```

SPJA queries are the most general and the most difficult queries, and only Count aggregation has been studied in the literature. Thus, in this article, we only discuss Count aggregation when SPJA queries are concerned.

Query type		State-of-the-art Result	
		Tuple-DP	User-DP
Count /Sum	SA	[24]: $O(1)$-worst-case optimal	
	Self-join -free SJA	[19, 20]: $\big(O(1), O(1)\big)$-neighborhood optimal	[21]: $\big(1, O(1)\big)$-down neighborhood optimal
	Self-join SJA		[16]: $\big(1, \tilde{O}(1)\big)$-down neighborhood optimal
			[25]: $\big(1, \tilde{O}(1)\big)$-down neighborhood optimal
	SPJA	[19, 20]: No optimal guarantee	[16] and [25]: No optimal guarantee
		[25]*:$\big(\tilde{O}(1), \tilde{O}(1)\big)$-downward neighborhood optimal	
Max	SA	[21]: $\big(\tilde{O}(1), 2\big)$-down neighborhood optimal	
	SJA	[25]: $\big(\tilde{O}(1), 2\big)$-down neighborhood optimal	

Table 1: Summary of state-of-the-art results for answering SQL queries under tuple-DP and user-DP. All results achieve ε-DP and have polynomial running time except *.

1.3 Optimality Measures for Utility

As the output of a DP mechanism must be randomized, we often use a constant-probability error bound to measure its utility:

$$\mathrm{Err}(M_Q, \mathbf{I}) = \inf\Big\{\xi : \Pr\left[\|M_Q(\mathbf{I}) - Q(\mathbf{I})\| \le \xi\right] \ge 2/3\Big\}.$$

Most classical DP mechanisms are based on the notion of *sensitivity* of the query Q. First, the *local sensitivity* of Q at instance $\mathbf{I}$ is how much $Q(\mathbf{I})$ can change when $\mathbf{I}$ changes to one of its neighbors, i.e.,

$$\mathrm{LS}_Q(\mathbf{I}) = \sup_{\mathbf{I}', \mathbf{I} \sim \mathbf{I}'} \|Q(\mathbf{I}) - Q(\mathbf{I}')\|.$$

The *global sensitivity* of Q is

$$\mathrm{GS}_Q = \sup_{\mathbf{I}} \mathrm{LS}_Q(\mathbf{I}).$$

For a 1-dimensional query $Q : \mathcal{I} \to \mathbb{R}$, adding a Laplace noise proportionate to $\mathrm{GS}_Q/\varepsilon$ preserves ε-DP. This mechanism is referred to as the *Laplace mechanism*, which yields an error of $O(\mathrm{GS}_Q)$[2]. A simple case that can be handled by the Laplace mechanism is any SA-`Count` query under tuple-DP, for which we have $\mathrm{GS}_Q = 1$.

Worst-case optimality.

The classical optimality notion is *worst-case optimality*. Let $\mathcal{M}_Q$ be the class of all (ε, δ)-DP mechanisms for query Q. The worst-case lower bound is

$$\mathcal{L}_{\mathrm{wst}} = \inf_{M'_Q \in \mathcal{M}_Q} \sup_{\mathbf{I}' \in \mathcal{I}} \mathrm{Err}(M'_Q, \mathbf{I}').$$

It can be shown that for any constant ε, $\mathcal{L}_{\mathrm{wst}} \ge \mathrm{GS}_Q/2$ for any Q. So the Laplace mechanism is already worst-case optimal, not just for SA-`Count` queries, but for all Q. However, the Laplace mechanism is hardly a satisfactory, or even valid, solution to any query other than SA-`Count`, as GS_Q is often large or unbounded for many Q. Consider the SJA-`Count` query given at the beginning of the article. We can construct an $\mathbf{I}$ in which there is just one supplier and one customer, while all lineitems are shipped from this supplier to this customer. Then we delete the customer to obtain $\mathbf{I}'$ (for user-DP, we also need to delete all the lineitems). Such a pair of neighboring instances imply that $\mathrm{GS}_Q = \infty$, so the Laplace mechanism cannot be applied. One may artificially impose a limit on GS_Q, but this is not a theoretically elegant solution; in practice, this is not satisfying, either, since this limit must be set *a priori*, so is often conservatively large.

Instance optimality.

The failure of the Laplace mechanism means that some instance-specific optimality should be employed. The strongest such notion is *instance optimality*. More precisely, let

$$\mathcal{L}_{\mathrm{ins}}(\mathbf{I}) := \inf_{M'_Q \in \mathcal{M}_Q} \mathrm{Err}(M'_Q, \mathbf{I})$$

be the smallest error any $M'_Q \in \mathcal{M}_Q$ can achieve on $\mathbf{I}$. Then an DP mechanism M_Q is *c-instance optimal* if $\mathrm{Err}(M_Q, \mathbf{I}) \le c \cdot \mathcal{L}_{\mathrm{ins}}(\mathbf{I})$ for every $\mathbf{I}$, where c is called the *optimality ratio*. Unfortunately, for every $\mathbf{I}$, one can design a trivial $M'_Q(\cdot) \equiv Q(\mathbf{I})$ that has 0 error on $\mathbf{I}$ (but fails miserably on other instances), so $\mathcal{L}_{\mathrm{ins}}(\cdot) \equiv 0$, which rules out instance-optimal DP mechanisms.

Neighborhood optimality.

As instance optimality is unattainable, [8, 20] consider a relaxed version of instance optimality where we compare M_Q against any M'_Q that is required to work well not just on $\mathbf{I}$, but also on its k-neighbors, i.e., instances within distance[3] k from $\mathbf{I}$. More precisely, we

[2]The O notation omits the dependency on ε and log log factors, and the $\tilde{O}$ notation further omits polylogarithmic factors.

[3]Distance between $\mathbf{I}$ and $\mathbf{I}'$ is the number of individuals'

define the target error on $\mathbf{I}$ as

$$\mathcal{L}_{\text{nbr}}(\mathbf{I}, k) := \inf_{M'_Q \in \mathcal{M}_Q} \sup_{\mathbf{I}': d(\mathbf{I}, \mathbf{I}') \le k} \text{Err}(M'_Q, \mathbf{I}').$$

Then, M_Q is (k, c)-*neighborhood optimal* if $\text{Err}(M_Q, \mathbf{I}) \le c \cdot \mathcal{L}_{\text{nbf}}(\mathbf{I}, k)$ for every $\mathbf{I}$. Note that neighborhood optimality interpolates between instance optimality ($k = 0$) and worst-case optimality ($k = \infty$), with smaller values of k corresponding to stronger optimality.

Neighborhood optimality has been adopted for analyzing DP mechanisms for certain machine learning problems [8] and SJA-COUNT queries under tuple-DP [20]. However, it degenerates into worst-case optimality (for any $k \ge 1$), hence meaningless, when the query has a MAX or SUM aggregation. Consider an SA-MAX query that returns the highest salary of any customer. For every $\mathbf{I}$, we can construct a neighboring instance $\mathbf{I}'$ by adding a customer with an arbitrarily high salary, implying $\text{LS}_Q(\mathbf{I}) = \infty$. Vadhan [46] shows that $\mathcal{L}_{\text{nbr}}(\mathbf{I}, 1) \ge \text{LS}_Q(\mathbf{I})/2$, so $\mathcal{L}_{\text{nbr}}(\mathbf{I}, 1)$ is also unbounded. A similar construction works for a SA-SUM query or an SJA-COUNT query under user-DP, by just adding a user contributing to arbitrarily many tuples.

The reason why neighborhood optimality fails to work in these cases is that we require M'_Q to work well on any neighbor $\mathbf{I}'$ of $\mathbf{I}$. For these queries, there always exists a bad neighbor that contains a heavy contributor. This is too high a requirement for M'_Q, hence too low an optimality notion for M_Q.

Down-neighborhood optimality.

To address the issue, Dong et al. [16] revised $\mathcal{L}_{\text{nbr}}(\cdot, \cdot)$ to

$$\mathcal{L}_{\text{d-nbr}}(\mathbf{I}, k) := \min_{M'_Q \in \mathcal{M}_Q} \max_{\mathbf{I}': d(\mathbf{I}, \mathbf{I}') \le k, \mathbf{I}' \subseteq \mathbf{I}} \text{Err}(M'_Q, \mathbf{I}'),$$

namely, we require M'_Q to work well only on $\mathbf{I}'$ and its k-*down-neighbors*, which can be obtained by removing at most k users' data from $\mathbf{I}$. Then (k, c)-*down neighborhood optimality* is defined analogously. The $k = 1$ case is of particular interest, as it can be shown that $\text{DS}_Q(\mathbf{I}) \ge \mathcal{L}_{\text{nbr}}(\mathbf{I}, 1) \ge \text{DS}_Q(\mathbf{I})/2$ where $\text{DS}_Q(\mathbf{I})$ is the *downward local sensitivity* of Q at $\mathbf{I}$:

$$\text{DS}_Q(\mathbf{I}) = \max_{\mathbf{I}', \mathbf{I} \sim \mathbf{I}', \mathbf{I}' \subseteq \mathbf{I}} \big\| Q(\mathbf{I}) - Q(\mathbf{I}') \big\|.$$

Thus, we can use $\text{DS}_Q(\mathbf{I})$ as a proxy to prove down neighborhood optimality, which is easier and has simple interpretations for many queries. For example, for a query Q with COUNT or SUM aggregation, $\text{DS}_Q(\mathbf{I})$ is maximum user contribution in $Q(\mathbf{I})$. For an SA-MAX query, $\text{DS}_Q(\mathbf{I})$ is the gap between the maximum value and the second maximum value. Unlike $\text{LS}_Q(\mathbf{I})$, $\text{DS}_Q(\mathbf{I})$ is always bounded and usually small on most instances, so down-neighborhood optimality is more meaningful.

information they differ. Under user-DP, that refers to the number of different users while under tuple-DP, that represents the number of different tuples.

1.4 Overview of Results

Table 1 provides an overview of the state-of-the-art solutions for answering various queries under DP. Under tuple-DP, as mentioned, the Laplace mechanism [24] already achieves an error of $O(1)$ for SA-COUNT queries, which is $O(1)$-worst-case optimal. For SA-MAX queries, [21] achieves $\big(\tilde{O}(1)), 2\big)$-down neighborhood optimality. For SJA queries, [19, 20] achieve $\big(O(1), O(1)\big)$-neighbor -hood optimal error. For SJA queries with the MAX aggregation, while there is no dedicated work for this problem, the mechanism [25] designed for user-DP can be employed. This is possible because any user-DP mechanism can also handle tuple-DP, as mentioned in Section 1.1. For SPJA queries, [20] currently delivers the best performance but still lacks an optimal error guarantee.

Moving towards user-DP, for SJA queries, when the queries are self-join-free, [21] achieves $\big(1, O(1)\big)$-down neighborhood optimal error. For self-join queries, both [16] and [25] achieve $\big(1, \tilde{O}(1)\big)$-down neighborhood optimal error and the log factors hidden by $\tilde{O}$ in these solutions are not comparable. For SJA queries with the MAX aggregation, [25] achieves the $\big(\tilde{O}(1), 2\big)$-downward neighborhood optimal error. For SPJA queries, both [16] and [25] are applicable, yet neither guarantees optimal utility. Meanwhile, [25] gives another solution achieving $\big(\tilde{O}(1), \tilde{O}(1)\big)$-downward neighborhood optimal error while taking super-polynomial time. Throughout the article, we use *data complexity* [2] when talking about running times, i.e., the running time is measured as a function of the database size N, while the query size (i.e., number of relations and attributes in the query) is considered a constant. All the aforementioned solutions have polynomial running times, except for the last one. Furthermore, all of these solutions achieve pure-DP.

2. DP PROPERTIES

The following properties of DP will be useful:

LEMMA 1 (POST PROCESSING [24]). *If $M_{Q_1} : \mathcal{I} \to \mathcal{Y}$ satisfies (ε, δ)-DP and $M_{Q_2} : \mathcal{Y} \to \mathcal{Z}$ is any randomized mechanism, then $M_{Q_2}(M_{Q_1}(\mathbf{I}))$ satisfies (ε, δ)-DP.*

LEMMA 2 (COMPOSITION THEOREM [24]). *If M_Q is an adaptive composition[4] of differentially private mechanisms $M_{Q_1}, \ldots, M_{Q_k}$, where each M_{Q_k} satisfies (ε, δ)-DP, then M satisfies (ε', δ')-DP, where*

1. *$\varepsilon' = k\varepsilon$ and $\delta' = k\delta$; [Basic Composition]*
2. *$\varepsilon' = \varepsilon\sqrt{2k \log \frac{1}{\delta''}} + k\varepsilon(e^{\varepsilon} - 1)$ and $\delta' = k\delta + \delta''$ for any $\delta'' > 0$. [Advanced Composition]*

LEMMA 3 (GROUP PRIVACY [24]). *If M_Q is an $(\varepsilon_0, \delta_0)$-DP mechanism, then for any two instances $\mathbf{I}, \mathbf{I}'$ with $d(\mathbf{I}, \mathbf{I}') = \lambda$, M_Q satisfies $(\lambda\varepsilon_0, \lambda e^{\lambda\varepsilon_0}\delta_0)$-DP.*

[4] Adaptive composition refers to a sequence of mechanisms, where the choice of each mechanism can depend on the outcomes of the previous mechanisms.

LEMMA 4 (PARALLEL COMPOSITION [36]). *If M_{Q_1}, M_{Q_2} satisfy ε_1-DP and ε_2-DP, and $\mathcal{X}_1, \mathcal{X}_2 \subseteq \mathcal{X}$ are two disjoint input domains, then $\big(M_{Q_1}(\mathbf{I} \cap \mathcal{X}_1), M_{Q_2}(\mathbf{I} \cap \mathcal{X}_2)\big)$ satisfies $\max(\varepsilon_1, \varepsilon_2)$-DP.*

3. QUERY EVALUATION UNDER TUPLE-DP

3.1 SA Queries

As mentioned, SA-COUNT queries can be readily handled by the Laplace mechanism. SA-SUM queries can be handled as self-join-free SJA-COUNT queries under user-DP, which will be discussed in Section 4.1. This section thus only discusses SA-MAX queries. In this problem, $\mathbf{I}$ can be regarded as a multiset of integers $\{x_1, x_2, \ldots, x_N\}$, we reorder $\mathbf{I}$ such that $x_1 \leq \cdots \leq x_N$, and our goal is to design a DP mechanism M_Q such that

$$x_{N-\rho} \leq Q(\mathbf{I}) \leq x_N.$$

Such a guarantee is said to have a *rank error* of ρ. Note that this is equivalent to $(\rho, 2)$-down neighborhood optimality.

For this problem, Asi and Duchi introduced the *inverse sensitivity mechanism*, which instantiates the *exponential mechanism* [24], a fundamental ε-DP framework. The inverse sensitivity mechanism operates based on an assumption of data boundedness within the range $[0, U]$ and achieves ε-DP while maintaining a rank error of $O(\log(U))$.

Later, Huang et al. [26] proposed an alternative mechanism by transforming the maximum problem into a counting problem. This algorithm also requires a similar assumption of data boundedness as the inverse sensitivity mechanism. The high-level idea is to find the largest r such that $[r, U]$ contains more than c elements, where c is a predetermined parameter. By employing a binary search, the desired r can be located using $\log(U)$ counting queries. The composition theory is then utilized to allocate the privacy budget for each counting query. Their mechanism is under the *concentrated differential privacy* (CDP) [11], which is a DP notation between ε-DP and (ε, δ)-DP. By carefully selecting the parameter c, they achieve a rank error of $O\big(\sqrt{\log(U)\log\log(U)}\big)$. Moreover, there exists a corresponding ε-DP version of this mechanism by replacing the Gaussian mechanism with the Laplace mechanism. However, this version yields a higher rank error of $O\big(\log(U)\log\log(U)\big)$ than the inverse sensitivity mechanism.

Dong and Yi [21] also reduced the problem to a counting problem. However, they took a different route: Instead of seeking an interval $[r, U]$, they identify the smallest value of r such that the interval $[0, r]$ encompasses the majority of elements. More precisely, they iteratively set $r = 1, 2, \ldots$ and inquire whether $[0, r]$ contains more than $N - c$ data, where c is another predefined parameter. When applying the composition theory, this method has a large error for each counting query, further leading to a large rank error. Instead, they use the *sparse vector technique* [23]. The algorithm first uses constant noise to obscure the threshold $N - c$. During each iteration, the constant noise is added to mask the counter result for $[0, r]$, and this noisy counter is then compared with the noisy threshold. Finally, the algorithm returns the first r for which its noisy counter surpasses the noisy threshold. The entire process can be shown to preserve ε-DP. In comparison to binary search, this approach doesn't require the division of privacy budget but generates more noisy outcomes. Additionally, the boundedness assumption becomes unnecessary. Through careful selection of $c = \tilde{O}(1)$, the algorithm achieves an instance-specific rank error of $O(\log(x_{\text{Max}}))$, where x_{Max} is the maximum value in $\mathbf{I}$.

3.2 SJA Queries

Let us begin by discussing JA queries. Joins make the problem more challenging, as a single tuple can now influence numerous join results, and the global sensitivity becomes ∞. A relatively easy approach is to add constraints so as to reduce global sensitivity. McSherry [36] solves the problem by restricting to one-to-one joins. Proserpio et al. [42] propose wPINQ to extend the work of McSherry to support general equijoins: by assigning weights to tuples and scaling down the weights, their algorithm ensures each tuple can at most affect one on final counting result. However, this only works well when one tuple affects a fixed number of results. Palamidessi and Stronati [41] add constraints on the attribute range. Arapinis et al. [7] and Narayan et al. [37] consider functional dependencies and cardinality constraints.

Another way to deal with the issue of a high global sensitivity is tempting to use the sensitivity of the query on the particular given instance like local sensitivity $\text{LS}_Q(\mathbf{I})$. However, as pointed out by Nissim et al. [39], using the local sensitivity to calibrate noise is not DP. This is because the local sensitivity can be very different on two neighboring databases, so the noise level may reveal information about an individual tuple. Essentially, the problem is that local sensitivity, when considered as a query, has high global sensitivity.

To get around the problem, the idea is to use a smooth (i.e., having low global sensitivity) upper bound of the local sensitivity, named *smooth sensitivity* (SS_Q) [39]. Similar to local sensitivity, smooth sensitivity is also instance-dependent and usually can be much smaller than global sensitivity. But different from local sensitivity, it eliminates abrupt changes between neighboring instances, hence the name "smooth sensitivity". More precisely, for any $\mathbf{I} \sim \mathbf{I}'$, we have $\text{SS}^\beta_Q(\mathbf{I}) \leq \text{SS}^\beta_Q(\mathbf{I}) \cdot e^\beta$. By selecting β to be $\Theta(\varepsilon)$ and incorporating noise sampled from a general Cauchy distribution[5], scaled by $\text{SS}^\beta_Q(\mathbf{I})$, we get an ε-DP mechanism. Furthermore, it can be demonstrated that smooth sensitivity achieves $(O(1), O(1))$-neighborhood optimal error for answering multi-way join counting queries under tuple-DP [20]. However, computing the smooth sensitivity by defini-

[5]The general Cauchy distribution has pdf $h(z) \propto \frac{1}{1+|z|^\gamma}$.

tion in general takes exponential time. Dong and Yi devised a method to reduce its computational cost for multi-way join counting queries to $N^{O(\log N)}$, which is still super-polynomial.

Due to the absence of an efficient algorithm for calculating the smooth sensitivity for multi-way join counting queries under tuple-DP, Johnson et al. [28] introduced *elastic sensitivity*. Elastic sensitivity is an approximation of smooth sensitivity while preserving its "smoothness property". In contrast to smooth sensitivity, elastic sensitivity can be computed in linear time. However, it lacks any utility guarantee. Theoretically, the gap between elastic sensitivity and smooth sensitivity can be as large as $O\left(N^{n-1}\right)$.

To tackle this challenge, Dong and Yi proposed *residual sensitivity* [19, 20], which is another valid approximation of smooth sensitivity. For utility, residual sensitivity is a constant-factor upper bound on smooth sensitivity, which can be used to add noise, resulting in an $(O(1), O(1))$-neighborhood optimal error. In terms of efficiency, residual sensitivity can be computed through a constant number of AJAR/FAQ queries [27, 4] with $\tilde{O}(1)$ additional computations. Each AJAR/FAQ query can be processed within $O(N^w)$ time [40, 9], where w is its AJAR/FAQ width, a constant depending on the query only.

Then, let us consider the selection operations. The traditional approach to dealing with selection operation [34, 28, 19] is to evaluate the query with selection but compute the sensitivity without considering the selection conditions. This yields a valid DP mechanism but loses optimality. To see this, just consider an extreme case where a selection condition always returns False. Then the query becomes a trivial query and the optimal (under any notion of optimality) mechanism is $M(\cdot) \equiv 0$, i.e., $\mathrm{Err}(M, \mathbf{I}) = 0$ for all $\mathbf{I}$, but the sensitivity of the query without the selection condition must be nonzero. Meanwhile, Dong and Yi demonstrated how to extend residual sensitivity to incorporate selection operations while maintaining its neighborhood optimality [20]. Additionally, when all selection conditions are inequalities and comparisons, the algorithms can still be run in polynomial time.

3.3 SPJA Queries

The conventional approach for answering SPJA queries under tuple-DP simply disregards the projection. Dong and Yi extended residual sensitivity to more effectively handle projection so as to reduce the noise [20]. This extended algorithm can also be executed using a constant number of AJAR/FAQ queries. However, it does not have neighborhood optimality.

4. QUERY EVALUATION UNDER USER-DP

Let us now move towards the user-DP. As mentioned, user-DP exclusively focuses on join queries, and self-joins can bring unique challenges. In this section, we first review the works for self-join-free queries. Subsequently, we delve into the techniques employed to handle self-joins. Next, we talk about how to answer SJA-Max queries and then SPJA queries.

4.1 Self-join-free SJA Queries

As mentioned, the challenge that arises with user-DP compared to tuple-DP is that each individual can own arbitrarily many tuples. Consider the self-join-free SJA query introduced in Section 1.2, where we count items while protecting the privacy of customers. In this context, a customer could theoretically possess an unbounded number of items and adding such a customer to the database can cause an unbounded change in the query result. A simple fix is to assume a finite GS_Q, which can be justified in practice because we may never have a customer with, say, more than a million items. However, assuming such a GS_Q limits the allowable database instances, one tends to be conservative and sets a large GS_Q. This allows the Laplace mechanism to work, but adding noise of this scale clearly eliminates any utility of the released query answer. Furthermore, it is clear that this issue persists across all instances, leading to $\mathrm{LS}_Q(\mathbf{I}) \equiv \mathrm{GS}_Q = \infty$. This means the sensitivity-based techniques used for answering SJA queries under tuple-DP will lose utility since those sensitivity measures are all upper bounds of local sensitivity.

The issue above was first identified by Kotsogiannis et al. [34], who also formalized the user-DP. Their solution is the *truncation mechanism*, which simply deletes all customers with more than τ items before applying the Laplace mechanism, for some threshold τ. After truncation, the query has sensitivity τ, so adding noise of scale τ is sufficient. Such an idea has also been used in a later work [45]. A well-known issue for the truncation mechanism is the bias-variance trade-off: In one extreme $\tau = \mathrm{GS}_Q$, it degenerates into the naive Laplace mechanism with a large noise (i.e., large variance); in the other extreme $\tau = 0$, the truncation introduces a bias as large as the query answer. Both [34] and [45] use a heuristic approach to find such a τ, without offering any optimal guarantee.

As mentioned, self-join-free SJA queries under user-DP are equivalent to the sum estimation problem, and the issue of how to choose a near-optimal τ has been extensively studied in the statistics and machine learning community [1, 5, 6, 26, 21]. In this context, $\mathbf{I}$ is treated as an ordered multiset of integers $x_1, x_2, \ldots, x_N$, where each x_i corresponds to an individual user's contribution to $Q(\mathbf{I})$, and $Q(\mathbf{I}) = \sum_i x_i$. The truncation mechanism is to delete those $x_i > \tau$.[6] Furthermore, it is clear that x_N is the maximum user contribution, i.e., $\mathrm{DS}_Q(\mathbf{I})$. An observation is that by setting $\tau = x_N$, we can eliminate bias and introduce noise at a scale of $O(\mathrm{DS}_Q(\mathbf{I}))$, thereby achieving $(1, O(1))$-down neighborhood optimal error. Then, the problem is reduced to estimate x_N. The discussion of works on estimating x_N under DP can be found in Section 3.1, where the state-of-the-art algorithm yields a rank error of $O(\log(x_N))$. Using such

[6]Some works use clipping instead of truncation, i.e., clipping x_i to τ if $x_i > \tau$. Since will not affect the result asymptotically, we will use the truncation mechanism in our discussion.

a noisy estimated x_N as the truncation threshold leads to an error of $O\Big(\mathrm{DS}_Q(\mathbf{I}) \cdot \log\big(\mathrm{DS}_Q(\mathbf{I})\big)\Big)$.

Can we further enhance this result? Recall that the $O(\log(x_N))$ rank error has already been proven to be optimal, so it seems that this outcome can't be improved if we employ x_N as the truncation threshold. However, for the maximum problem, our focus is solely on achieving rank error, with the aim of avoiding relative error. Astute readers would recognize that when x_N serves as the truncation threshold, we essentially need only a constant approximation of x_N. In other words, we can tolerate some relative error in locating such x_N.

Dong and Yi [21] leveraged this finding to devise a mechanism for identifying a τ such that $x_{N-k} \le \tau \le 2x_N$. By permitting relaxation on the upper boundary, they manage to reduce the rank error from $O(\log(x_N))$ to $O(\log\log(x_N))$. Implementing such a τ results in an error of $O\Big(\mathrm{DS}_Q(\mathbf{I}) \cdot \log\log\big(\mathrm{DS}_Q(\mathbf{I})\big)\Big)$ in the sum estimation, which is $\Big(1, O\big(\log\log(\mathrm{DS}_Q(\mathbf{I}))\big)\Big)$-down neighborhood optimal. Furthermore, [21] points out the optimality ratio $O\big(\log\log(\mathrm{DS}_Q(\mathbf{I}))\big)$ cannot be improved.

4.2 SJA Queries with Self-joins

When addressing SJA queries under user-DP, the presence of self-joins brings another challenge. Specifically, all the aforementioned techniques for selecting a truncation threshold τ heavily depend on the assumption of individual independence, i.e., the addition or removal of one individual doesn't impact the data of another individual. However, this assumption no longer holds when the query involves self-joins. In fact, the truncation mechanism itself falls as illustrated in the following example.

Using the query provided at the beginning of the article as an example, let us assume that both `Customer` and `Supplier` are primary private relations. This is like an edge counting query in a bipartite graph, where the nodes on the left side represent suppliers, the nodes on the right side represent customers, and the edges represent items. With a given truncation threshold τ, we intend to eliminate all nodes with degrees higher than τ. To illustrate a failure of the truncation mechanism, let us construct an instance $\mathbf{I}$ as follows: each customer only purchases a single item, while each supplier provides τ items. In other words, each left-side node has a degree of τ, while each right-side node has a degree of 1. In total, there are N items. Now, consider a neighboring instance $\mathbf{I}'$, which we create by inserting a right-side node that is connected to every existing left-side node. In $\mathbf{I}'$, every left-side node has a degree of $\tau + 1$. When we do truncation by τ, the truncated result for $\mathbf{I}$ is N, whereas for $\mathbf{I}'$, it is 0 due to the truncation of all left-side nodes. Adding noise at a scale of τ cannot obscure their difference, consequently violating the DP.

The truncation mechanism fails because, after truncation, the query's sensitivity is no longer bounded by τ. More fundamentally, this is due to the correlation among the individuals introduced by self-joins. In the example above, we see that the addition of one node may cause the degrees of many others to increase. For the problem of graph pattern counting under node-DP, which can be formulated as a self-join SJA query under user-DP as previously mentioned, Kasiviswanathan et al. [31] proposed a linear program (LP)-based truncation mechanism to fix the issue. Dong et al. [16] then extended this solution to support general SJA queries.

Now, the remaining task is how to determine the appropriate τ when dealing with self-joins. On one hand, [31] does not study the selection of τ for graph pattern counting queries, rendering their mechanism devoid of any utility guarantee. On the other hand, adopting a similar approach to selecting τ as self-join-free queries does not work. This is also because a single individual can influence the contributions of numerous others. In the above example, all suppliers contribute τ in $\mathbf{I}$, while in $\mathbf{I}'$ each supplier contributes $\tau + 1$. Running a DP algorithm to estimate the maximum contribution is likely to yield τ and $\tau + 1$ for $\mathbf{I}$ and $\mathbf{I}'$ respectively, making them distinguishable.

To address this challenge, Dong et al. proposed *Race-to-the-Top* (R2T) [16] for adaptively choosing τ in combination with any valid DP truncation mechanism that satisfies specific properties: (1) $Q(\mathbf{I}, \tau) \le Q(\mathbf{I})$; (2) the sensitivity of $Q(\mathbf{I}, \tau)$ is bounded by τ; (3) $Q(\mathbf{I}, \tau) = Q(\mathbf{I})$ for $\tau \ge \mathrm{DS}_Q(\mathbf{I})$. Intuitively, such a $Q(\mathbf{I}, \tau)$ gives a stable (property (1)) underestimate (property (2)) of $Q(\mathbf{I})$, while reaches $Q(\mathbf{I})$ for τ sufficiently large (property (3)). Notably, $Q(\mathbf{I}, \tau)$ itself is not DP. Instead of directly determining τ, R2T directly provides a privatized query answer. The central idea is to try out $Q(\mathbf{I}, \tau)$ with $\tau = 2, 4, 8, \ldots, \mathrm{GS}_Q$, and somehow pick the "winner" of the race (the maximum) to estimate $Q(\mathbf{I})$. To ensure DP in this process, noise of $\mathrm{Lap}(\tau/\varepsilon')$ is added to each $Q(\mathbf{I}, \tau)$, requiring the privacy budget to be divided as $\varepsilon' = \varepsilon/\log(\mathrm{GS}_Q)$ since multiple $Q(\mathbf{I}, \tau)$ are attempted. Yet, this noise-masked $Q(\mathbf{I}, \tau)$ can turn out to be extremely uncertain and potentially much greater than $Q(\mathbf{I})$, particularly with a larger value of τ. To get out of this problem, we shift $Q(\mathbf{I}, \tau)$ down by an amount that roughly equals the scale of the noise, i.e., $\tilde{O}(\tau)$. This step ensures that the noise-masked $Q(\mathbf{I}, \tau)$ is generally underestimated compared to the true result $Q(\mathbf{I})$ thus we can pick the "winner" of the race. With this idea alongside the LP-based truncation mechanism, R2T achieves an error of $O\big(\mathrm{DS}_Q(\mathbf{I}) \cdot \log(\mathrm{GS}_Q(\mathbf{I})) \cdot \log\log(\mathrm{GS}_Q(\mathbf{I}))\big)$, which is indeed $\Big(1, O\big(\log(\mathrm{GS}_Q(\mathbf{I})) \cdot \log\log(\mathrm{GS}_Q(\mathbf{I}))\big)\Big)$-down neighborhood optimal. Additionally, for efficiency, the computation bottleneck of R2T is the $\log(\mathrm{GS}_Q)$ LPs, each contains $|J(\mathbf{I})|$ variables and $|J(\mathbf{I})| + N$ constraints, with $J(\mathbf{I})$ denotes join results, which is equivalent to $|Q(\mathbf{I})|$ for counting queries. Dong et al. also provide techniques to speed up this process and build a system employing PostgreSQL and the CPLEX LP solver.

Following R2T, Fang et al. [25] introduced another LP-based mechanism (to be discussed in detail in the upcoming section). This mechanism is also applicable

to answering SJA queries. Rather than an assumption of a fixed GS_Q, their algorithm necessitates a predefined output range $[1, R]$ and achieves an error of $O(\mathrm{DS}_Q(\mathbf{I}) \cdot \log(R))$. Notably, GS_Q must inherently be smaller than R, but the error of R2T possesses an additional loglog factor leading to these two upper bounds do not dominate each other. Additionally, their algorithm requires solving $\Theta(\log(R))$ LPs, each of which has a more complex structure than those used in R2T. The experiments show that both mechanisms yield comparable error levels, with R2T having significantly lower running times.

4.3 SJA Queries with Max Aggregation

Now, let us discuss SJA queries with Max aggregation. The objective of these queries is to identify the highest value among the aggregated attributes over the join results. This variation of the query introduces new challenges to achieving optimal utility and current techniques cannot effectively address them. These difficulties even exist in self-join-free queries. In this problem, one intuitive approach is to apply the truncation mechanism to establish an upper bound on the number of join results associated with each individual. Consequently, with the group privacy property of DP, we can transform this problem into a SA query with Max aggregation under tuple-DP. During the truncation phase, the optimal choice for τ is to use $\kappa(\mathbf{I})$, which is the maximum number of join results corresponding to a single user in $\mathbf{I}$. By allocating the privacy budget accordingly, the ultimate result has a rank error of $\tilde{O}(\kappa(\mathbf{I}))$.

However, despite that $\kappa(\mathbf{I})$ is the maximum number of join results corresponding to one user, a rank error of $\tilde{O}(\kappa(\mathbf{I}))$ doesn't necessarily imply $(\tilde{O}(1), c)$-down neighborhood optimality for arbitrary c. To see this, consider the query $R_1(\underline{A}) \bowtie R_2(A, B)$ that outputs the maximum value on attribute B, with relation R_1 designated as the primary private relation. The instance $\mathbf{I}$ contains two distinct user types. For the first $N/2$ users, each corresponds to a tuple (a_i) in R_1, and one tuple (a_i, N) in R_2. Meanwhile, for the remaining $N/2$ users, each corresponds to one tuple (a_i) in R_1 as well but $N/2$ tuples in R_2: $(a_i, 0), (a_i, 1), \ldots, (a_i, N-1)$. It is trivial to see that $Q(\mathbf{I}) = N$, and after removing any arbitrary k users where $k < N/2$, the query result will still remain unchanged at N. This implies that $\mathcal{L}_{\text{d-nbr}}(\mathbf{I}, k) = 0$ for $k < \frac{N}{2}$, as we can construct a mechanism M' where $M'(\cdot) \equiv N$. Nonetheless, it is clear that $\kappa(\mathbf{I}) = N$. Since there are only N join results with a value of N, a rank error of $\tilde{O}(\kappa(\mathbf{I}))$ means returning a value lower than N, which fails to achieve (d, c)-down neighborhood optimal error for any $d < \frac{N}{2}$ and any $c > 0$. Another approach is to transform the maximum problem into a counting problem, similar to what was discussed in Section 3.1. However, the $\tilde{O}(\mathrm{DS}_Q(\mathbf{I}))$ additive error of the counting problem would also lead to a $\tilde{O}(\kappa(\mathbf{I}))$ rank error. Acute readers would realize that the problem arises due to two main factors. First, the data distribution is skewed, implying that not all users in $\mathbf{I}$ correspond to $\kappa(\mathbf{I})$ join results. Second, we cannot guarantee that those high-value join results originate from the same individuals.

To address this issue, Fang et al. [25] have devised a general DP mechanism applicable to any monotonic query under the user-DP model called *ShiftedInverse*. Their algorithm requires a predefined output range $[1, R]$ and achieves a $\Big(O\big(\log(R)\big), O\big(\log(R)\big)\Big)$-down neighborhood optimal error. The high-level idea is, for each value $r \in [1, R]$, they determine $\mathrm{len}(\mathbf{I}, r)$, which is how many individuals should be excluded to achieve a query result less than r. Subsequently, they sample each r as an output with a probability proportional to $\mathrm{len}(\mathbf{I}, r)$. This process can be shown to satisfy the ε-DP under the user-DP while the challenge of guaranteeing down-neighborhood optimal error is to ensure that the output is underestimated. This can be achieved by "shifting" the target downward. More precisely, they use $\widehat{\mathrm{len}}(\mathbf{I}, r) = |\mathrm{len}(\mathbf{I}, r) - \Theta(\log(R))|$ in place of $\mathrm{len}(\mathbf{I}, r)$ during sampling. As a result, they will sample an r such that $\underline{\mathrm{len}(\mathbf{I}, r) = O(\log(R))$ with high probability, implying a $(O(\log(R)), O(\log(R)))$-down neighborhood optimal error$}$. Note that this underlined statement holds for arbitrary monotonic queries, while for specific queries, more favorable outcomes might be attainable. For instance, for maximum queries, this approach can lead to a $(O(\log(R)), 2)$-down neighborhood optimal error.

For self-join-free SJA queries with Max aggregation, all $\mathrm{len}(\mathbf{I}, r)$ values for $r \in [1, R]$ can be computed in linear time. However, for other functions like self-join SJA queries with Max and Sum aggregation, computing $\mathrm{len}(\mathbf{I}, r)$ requires a time complexity of $O(N^{\mathrm{len}(\mathbf{I}, r)})$. Even though, Fang et al. emphasize it is unnecessary to compute all $\mathrm{len}(\mathbf{I}, r)$ values. Nonetheless, implementing ShiftedInverse still requires a running time of $N^{O(\log(R))}$ in general, which is super-polynomial unless R is very small. To address this challenge, they propose an approximation of $\mathrm{len}(\mathbf{I}, \mathrm{r})$ specifically for certain functions. This modified mechanism, equipped with the approximate $\mathrm{len}(\mathbf{I}, \mathrm{r})$, is termed *approximate ShiftedInverse*. While approximate ShiftedInverse maintains ε-DP, the utility guarantee might not be upheld. When addressing self-join SJA queries with Max aggregation, each approximate $\mathrm{len}(\mathbf{I}, r)$ is formulated as an LP, and the approximate ShiftedInverse mechanism achieves a $(O(\log(R)), 2)$-down neighborhood optimal error. By the way, an approximation for SJA queries with Sum aggregation is also proposed to achieve the result mentioned in the last section.

4.4 SPJA Queries

For SPJA queries, first, since ShiftedInverse can handle arbitrary monotonic queries under user-DP, we can use it to answer SPJA queries and achieve $O\big(\log(R), \log(R)\big)$-down neighborhood error. However, in this case, ShiftedInverse cannot be computed in polynomial time. Additionally, Fang et al [25] also proposed an approximate ShiftedInverse mechanism for this problem, which can be computed using a logarithmic number of LPs.

However, this mechanism does not hold an optimal guarantee in utility. In parallel, Dong et al [16] proposed an LP-based truncation mechanism specifically designed for SPJA queries. This mechanism can also be integrated with R2T. However, their algorithm also lacks an optimal utility guarantee as well. Moreover, they presented a negative result indicating that achieving an error dependent on $\mathrm{DS}_Q(\mathbf{I})$ for answering SPJA queries under user-DP is unattainable. This essentially means that achieving a $(1, c)$-down neighborhood optimal error for any value of c is not achievable.

5. ANSWERING QUERIES UNDER DP IN MORE COMPLEX SETTING

All the aforementioned discussions consider the straightforward scenario of answering a single SQL query. Subsequently, more complex settings have been studied in the literature.

5.1 Multi-query Answering

In practice, queries often arrive in batches, thus it is natural to consider the multi-query problem in relational databases, which includes group-by queries as an important special case (i.e., each group corresponds to one query). For instance, if a data analyst is interested in the total number of items shipped for each date of the first month this year, s/he would issue the following query,

```
SELECT Lineitem.Shipdate, count(*)
FROM Customer, Orders, Lineitem
WHERE Lineitem.Shipdate > 2023-01-01
  AND Lineitem.Shipdate < 2023-04-30
  AND Customer.CK = Orders.CK
  AND Orders.OK = Lineitem.OK
GROUP BY Lineitem.Shipdate;
```

This query is equivalent to answering $d = 100$ SJA queries, with each query corresponding to a specific date. Let $\mathbf{Q} = (Q_1, \ldots, Q_d)$ represent the set of d queries that we aim to answer privately. We use the standard metric of root-mean-square error (RMSE), i.e., the ℓ_2 error to measure the utility.

The general approach to this multi-query problem is to use the privacy composition theory, i.e., we divide the privacy budget to the d queries and answer each query with the single-query mechanism. Using advanced composition, the utility suffers an $\tilde{O}(\sqrt{d})$-factor degradation which is the best we can if to answer SA queries with Count aggregation under tuple-DP. For more complex queries like SJA queries under user-DP, this method leads to an error of $\tilde{O}\big(\sqrt{d} \cdot \mathrm{DS}_{Q_k}(\mathbf{I})\big)$ for Q_k hence an RMSE of

$$\tilde{O}\left(\sqrt{d} \cdot \sqrt{\sum_{k=1}^{d} \mathrm{DS}_{Q_k}(\mathbf{I})^2}\right) \leq \tilde{O}\Big(d \cdot \mathrm{DS}_{\mathbf{Q}}(\mathbf{I})\Big).$$

However, this error is not optimal. An observation is that answering d self-join-free SJA queries under user-DP is equivalent to the sum estimation problem in d dimensions, where each user's contribution to these d queries can be seen as a vector, and the task is to compute their summation. This equivalence has an immediate consequence: the lower bound established for the sum estimation problem also leads to a lower bound for the multi-query problem, which is $\tilde{\Omega}\left(\sqrt{d} \cdot \mathrm{DS}_{\mathbf{Q}}(\mathbf{I})\right)$ [26, 29]. For self-join-free queries, [26] extends their algorithm designed for single self-join-free SJA query under user-DP to the multiple-query scenario. In their approach, they first estimate the maximum user contribution and employ that as the threshold for truncating heavy contributors. Subsequently, they use the Gaussian mechanism to add noise. Ultimately, they achieve an error of $\tilde{O}(\sqrt{d} \cdot \mathrm{DS}_{\mathbf{Q}}(\mathbf{I}))$. However, their mechanism requires a predefined $\mathrm{GS}_{\mathbf{Q}}$. On the other hand, Dong et al. [18] extended the 1-dimensional mechanism from [21] to d dimensions. This extension enables them to also achieve the optimal error of $\tilde{O}(\sqrt{d} \cdot \mathrm{DS}_{\mathbf{Q}}(\mathbf{I}))$ without the need for a predefined $\mathrm{GS}_{\mathbf{Q}}$.

For self-joins, akin to the single query case, the truncation mechanism encounters problems. Additionally, the LP-based mechanisms [16, 25] fundamentally do not work for multiple queries, as LP optimization is limited to one-dimensional queries. Dong et al. [18] also highlight that the straightforward extension of these LP-based approaches also does not work. Thereafter, they proposed an alternative approach to the multi-query problem. The initial version of their algorithm has an exponential running time, but they subsequently reduce it to polynomial time using quadratically constrained quadratic programming (QCQP), which can be computed in polynomial time. This novel approach enables them to achieve an error of $\tilde{O}(\sqrt{d} \cdot \mathrm{DS}_{\mathbf{Q}}(\mathbf{I}))$, matching the lower bound up to polylogarithmic factors.

Furthermore, Cai et al. [12] addressed the multi-query problem using an alternative approach. They generated a synthetic relational database under DP and used that to answer the subsequent queries. This methodology provides the advantage of accommodating a wide array of query types while maintaining error independence from the query dimensions. However, their proposed algorithm lacks any utility guarantee and only performs well when the domain of attributes is small.

5.2 Continual Observation

Data is seldom static. When private data evolves over time, there is a need to continually release sanitized query results about the data while preserving the privacy of the users who contribute to the data. This is precisely the problem of *continual observation under differential privacy*, introduced in the pioneering work of Dwork et al. [22]. Here, time is divided into discrete steps, and data is modeled as a (possibly infinite) stream of tuples arriving over time, one per time step and we release the query result after each time step. In the dynamic setting, Dwork et al. [22] proposed two natural DP definitions: In *event-DP*, two streams are neighbors if one can be obtained from the other by removing one item. In *user-DP*, each tuple is associ-

Query type		Current Result	
		Event-DP	User-DP
Count/Sum	SA	✓ [22], [13]	
	Self-join-free SJA	Open question	✓ [17]
	Self-join SJA		
	SPJA		
Max	SA	✓ [22], [13]	
	SJA	Open question	× [17]

Table 2: Whether the same asymptotic error as that in the static setting can be achieved with a continual observation setting.

ated with a user, and two streams are neighbors if one can be obtained from the other by removing all or any subset of items associated with one user. Clearly, the event-DP/user-DP in the dynamic setting aligns with the tuple-DP/user-DP in the static setting. Furthermore, under event-DP/user-DP, the problem at each given moment can be viewed as a static problem under tuple-DP/user-DP. Our target is to achieve the same error as the static setting each time. Up to now, this objective has been successfully realized for certain specific SQL queries.

The major result in [22] on event-DP is a black-box reduction to the static problem with only a $\mathrm{poly}\log(T)$-factor increase in the error for any *union-preserving* query Q, where T is an upper bound on the stream length. Chan et al. [13] extend this result to infinite streams, with the $\mathrm{poly}\log(T)$ factor replaced by $\mathrm{poly}\log(t)$, where t is the current length of the stream. A union-preserving query Q is one such that $Q(\mathbf{I}\cup\mathbf{I}') = Q(\mathbf{I}) + Q(\mathbf{I}')$ for any $\mathbf{I}, \mathbf{I}'$. Most natural functions (e.g., count, sum, max) are union-preserving. The high-level idea is to build a binary decomposition over all the T time steps $\log T$ levels of intervals and the DP mechanism for the static problem is invoked on each interval to return a noisy query result. Then the query result at any time can be obtained by at most $\log T$ such noisy results, one from each level. To set the privacy budgets of these intervals, it suffices to allocate $\varepsilon/\log T$ to each interval by basic composition (across levels) and parallel composition (within a level).

By applying state-of-the-art algorithms for static count, sum, and max queries within this framework, for any time t, we achieve an error of $O(\log^{1.5} t)$ error for count queries, an error of $O(x^t_{\max} \cdot \log\log(x^t_{\max}) \cdot \log^2 t)$ for sum queries, and a rank error of $O(\log(x^t_{\max}) \cdot \log^2 t)$ for max queries, where $x^t_{\max}$ is the maximum value of the instance at time t. These errors correspond to $\tilde{O}(1)$-worst-case optimal error, $\big(1, \tilde{O}(1)\big)$-down neighborhood optimal error, and $\big(\tilde{O}(1), 2\big)$-down neighborhood error for SA queries with Count, Sum, and Max aggregation, respectively. It is worth mentioning that, in the dynamic setting, Sum queries under event-DP differ from those under user-DP, thus needing separate consideration for event-DP.

When moving towards user-DP, one natural idea is to truncate the user contributions. More precisely, given some truncation threshold τ, we only retain the first τ items from each user. Then, we can use group privacy to divide the privacy budget and call the mechanism for event-DP. However, this approach encounters two key issues: Estimating a good τ requires a strong prior knowledge, which is impossible for infinite time domain cases; This leads to a non-time-specific error, i.e., errors across all time steps share the same dependency on τ. To address this issue, Dong at el. [17] used a dynamic τ to constrain user contributions. More precisely, they monitor the count of users with contributions surpassing τ and subsequently double τ when a sufficient number of users meet this criterion. Following each doubling iteration, the entire data stream is truncated using the updated τ and the DP mechanism over the truncated stream is also re-initialized. As a result, for sum queries, they match the error as the static setting up to polylogarithmic factors. For max queries, unfortunately, they show a negative result that no ε-DP mechanisms can achieve $(O(\sqrt{T}), c(T))$-down neighborhood optimality at each time, where T is the length of the stream and $c(T)$ is an arbitrary function of T. It is clear that under user-DP, self-join-free SJA queries with Sum and Max can be treated as a sum and max query since each join result only belongs to one user. We have corresponding positive and negative results for these two queries. For self-join queries, so far no known algorithm can handle them.

6. CONCLUSION

In this article, we have surveyed some recent results on query evaluation under differential privacy. There are two predominant DP policies in the relational model, namely tuple-DP and user-DP, and two instance-specific optimality notions, neighborhood optimality and down-neighborhood optimality. The choice of an appropriate optimality notion depends on the nature of the query under consideration and the DP policy adopted.

We conclude this article by mentioning two interesting directions for further investigation.

More Complex Scenarios.

As mentioned, the problems in the single-query setting have been reasonably well solved. However, there are many open questions in more complex scenarios, especially for multiple queries and under continuous observation. For multi-query answering, one open question is how to effectively handle queries involving max-

imum aggregation and projection operations. For continual observation, current studies are limited to queries involving single relations or their equivalent constructs, and how to handle join operations still remains an uncharted domain waiting for more exploration.

Integrating DP with Artificial Intelligence in a Relational Model.

Training machine learning models within the framework of a relational database has attracted lots of attention from the database community [44, 35, 3, 38, 32]. In this context, the training procedure operates over outcomes from join queries, with substantial efforts devoted to enhancing operational efficiency and curtailing storage overhead by avoiding explicit materialization of join results. Concurrently, how to integrate DP with the training of machine learning models over a flatted table has also been extensively studied. However, it remains an open question how to integrate DP into the training of machine learning models within a relational framework.

7. REFERENCES

[1] 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.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.

[3] M. Abo Khamis, H. Q. Ngo, X. Nguyen, D. Olteanu, and M. Schleich. In-database learning with sparse tensors. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 325–340, 2018.

[4] M. Abo Khamis, H. Q. Ngo, and A. Rudra. Faq: questions asked frequently. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 13–28, 2016.

[5] K. Amin, A. Kulesza, A. Munoz, and S. Vassilvtiskii. Bounding user contributions: A bias-variance trade-off in differential privacy. In *International Conference on Machine Learning*, pages 263–271. PMLR, 2019.

[6] G. Andrew, O. Thakkar, H. B. McMahan, and S. Ramaswamy. Differentially private learning with adaptive clipping. *arXiv preprint arXiv:1905.03871*, 2019.

[7] M. Arapinis, D. Figueira, and M. Gaboardi. Sensitivity of counting queries. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, 2016.

[8] H. Asi and J. C. Duchi. Instance-optimality in differential privacy via approximate inverse sensitivity mechanisms. *Advances in neural information processing systems*, 33, 2020.

[9] N. Bakibayev, T. Kociskỳ, D. Olteanu, and J. Závodnỳ. Aggregation and ordering in factorised databases. *Proceedings of the VLDB Endowment*, 6(14), 2013.

[10] 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, 2013.

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

[12] K. Cai, X. Xiao, and G. Cormode. Privlava: synthesizing relational data with foreign keys under differential privacy. *Proceedings of the ACM on Management of Data*, 1(2):1–25, 2023.

[13] T.-H. H. Chan, E. Shi, and D. Song. Private and continual release of statistics. *ACM Transactions on Information and System Security*, 2011.

[14] 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.

[15] T. Dick, C. Dwork, M. Kearns, T. Liu, A. Roth, G. Vietri, and Z. S. Wu. Confidence-ranked reconstruction of census microdata from published statistics. *Proceedings of the National Academy of Sciences*, 120(8):e2218605120, 2023.

[16] W. Dong, J. Fang, K. Yi, Y. Tao, and A. Machanavajjhala. R2T: Instance-optimal truncation for differentially privatequery evaluation with foreign keys. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2022.

[17] W. Dong, Q. Luo, and K. Yi. Continual observation under user-level differential privacy. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2190–2207. IEEE Computer Society, 2023.

[18] W. Dong, D. Sun, and K. Yi. Better than composition: How to answer multiple relational queries under differential privacy. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2023.

[19] W. Dong and K. Yi. Residual sensitivity for deferentially private multi-way joins. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2021.

[20] W. Dong and K. Yi. A nearly instance-optimal differentially private mechanism for conjunctive queries. In *Proc. ACM Symposium on Principles of Database Systems*, 2022.

[21] W. Dong and K. Yi. Universal private estimators. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 195–206, 2023.

[22] C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum. Differential privacy under continual observation. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 715–724, 2010.

[23] C. Dwork, M. Naor, O. Reingold, G. N. Rothblum, and S. Vadhan. On the complexity of differentially private data release: efficient algorithms and hardness results. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 381–390, 2009.

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

[25] J. Fang, W. Dong, and K. Yi. Shifted inverse: A general mechanism for monotonic functions under user differential privacy. 2022.

[26] Z. Huang, Y. Liang, and K. Yi. Instance-optimal mean estimation under differential privacy. *Advances in Neural Information Processing Systems*, 2021.

[27] M. R. Joglekar, R. Puttagunta, and C. Ré. Ajar: Aggregations and joins over annotated relations. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 91–106, 2016.

[28] 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.

[29] G. Kamath, J. Li, V. Singhal, and J. Ullman. Privately learning high-dimensional distributions. In *Proceedings of the 32nd Annual Conference on Learning Theory*, COLT '19, pages 1853–1902, 2019.

[30] V. Karwa, S. Raskhodnikova, A. Smith, and G. Yaroslavtsev. Private analysis of graph structure. *Proceedings of the VLDB Endowment*, 4(11):1146–1157, 2011.

[31] 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.

[32] M. A. Khamis, H. Q. Ngo, X. Nguyen, D. Olteanu, and M. Schleich. Learning models over relational data using sparse tensors and functional dependencies. *ACM Transactions on Database Systems (TODS)*, 45(2):1–66, 2020.

[33] D. Kifer and A. Machanavajjhala. No free lunch in data privacy. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 193–204, 2011.

[34] 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.

[35] A. Kumar, M. Boehm, and J. Yang. Data management in machine learning: Challenges, techniques, and systems. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1717–1722, 2017.

[36] 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, 2009.

[37] A. Narayan and A. Haeberlen. Djoin: Differentially private join queries over distributed databases. In *USENIX Symposium on Operating Systems Design and Implementation*, pages 149–162, 2012.

[38] M. Nikolic, H. Zhang, A. Kara, and D. Olteanu. F-ivm: learning over fast-evolving relational data. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2773–2776, 2020.

[39] 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.

[40] D. Olteanu and J. Závodnỳ. Size bounds for factorised representations of query results. *ACM Transactions on Database Systems (TODS)*, 40(1):1–44, 2015.

[41] C. Palamidessi and M. Stronati. Differential privacy for relational algebra: Improving the sensitivity bounds via constraint systems. In *QAPL*, 2012.

[42] D. Proserpio, S. Goldberg, and F. McSherry. Calibrating data to sensitivity in private data analysis. *Proceedings of the VLDB Endowment*, 7(8), 2014.

[43] P. Regulation. General data protection regulation. *Intouch*, 25:1–5, 2018.

[44] M. Schleich, D. Olteanu, and R. Ciucanu. Learning linear regression models over factorized joins. In *Proceedings of the 2016 International Conference on Management of Data*, pages 3–18, 2016.

[45] 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.

[46] S. Vadhan. The complexity of differential privacy. In *Tutorials on the Foundations of Cryptography*, pages 347–450. Springer, 2017.

[47] 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.