DOP-SQL: A General-purpose, High-utility, and Extensible Private SQL System

Jianzhe Yu
Hong Kong University of Science and Technology
jyuca@cse.ust.hk

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

Juanru Fang
Hong Kong University of Science and Technology
jfangad@cse.ust.hk

Dajun Sun
Hong Kong University of Science and Technology
dsunad@cse.ust.hk

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

ABSTRACT

Differential privacy (DP) has garnered significant attention from both academia and industry 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 SQLs while maintaining DP has become the holy grail in privacy-preserving query release.

In this demonstration, we present DOP-SQL, a DP SQL system that can answer a broad class of queries consisting of the selection, projection, aggregation, join, and group by operators. DOP-SQL has integrated a suite of down-neighborhood optimal DP mechanisms, thus achieving state-of-the-art utility. The current implementation of DOP-SQL is based on PostgreSQL, but its extensible feature allows it to be used in conjunction with any standard SQL engine.

PVLDB Reference Format:
Jianzhe Yu, Wei Dong, Juanru Fang, Dajun Sun, and Ke Yi. DOP-SQL: A General-purpose, High-utility, and Extensible Private SQL System. PVLDB, 17(12): 4385 - 4388, 2024.
doi:10.14778/3685800.3685881

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/hkustDB/DPSQL.

1 INTRODUCTION

As more personal data has been collected by companies, organizations, and governments, privacy concerns have become a major issue when it comes to how such data shall be used. Among the many privacy notions, *differential privacy (DP)* [7] has become the de facto standard that is now widely adopted in both government agencies and the industry. However, most existing DP systems only support reporting some simple aggregates over data stored in a single table (the notable example is the privatized aggregates published by the US Census Bureau). This severely limits the application scenarios of DP; ideally, data analysts would like to ask complicated, ad hoc analytical queries over personal data stored in a relational database using SQL. This would not only significantly broaden the scope of differential privacy, but also allows organizations to easily integrate this technology into their existing data systems, as SQL is still the dominant query language for data analytics with a large amount of legacy code.

The research community have studied the problem of how to answer general SQL queries under differential privacy extensively, but the existing DP SQL engines are not satisfactory. Table 1 provides a brief summary of existing systems, in comparison to DOP-SQL, which is our new DP SQL system to be demonstrated at the conference.

Level of privacy protection. Two DP policies for relational databases have been proposed, namely, *tuple-DP* and *user-DP*. As the names suggest, tuple-DP protects the privacy of tuples, while the latter protects the privacy of users. When each user possesses just one tuple, the two policies coincide, but in general, user-DP offers stronger privacy protection than tuple-DP. This is also the reason why most recent DP SQL systems, including DOP-SQL, have adopted user-DP. We provide more details on the DP policy in Section 2.1.

Query class. DOP-SQL supports any Selection-Projection-Join-Aggregation (SPJA) query, possibly with group-by, without any restrictions on the join type. The aggregation functions supported include all the standard ones such as `COUNT`, `COUNT DISTINCT`, `MAX/MIN`, `SUM`, and `QUANTILE`. In particular, the query class supported by DOP-SQL is a superset of all those supported by previous systems. Note that a query must have aggregations, as returning any original tuples from the database cannot possibly satisfy DP. Please see Section 2.2 for the detailed query syntax of DOP-SQL.

Utility. The last but most important aspect of DP SQL systems is utility, i.e., how close the privatized query answers are to the true answers. This is where DOP-SQL truly shines, as it is built under the framework of Down-neighborhood Optimal differentially Private mechanisms (hence the name DOP-SQL), which we detail in Section 2.4. Prior work [2, 4, 9] has conducted extensive experimentally evaluations comparing these DOP mechanisms with previous ones,

*Wei Dong is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:10.14778/3685800.3685881

Table 1: Comparison with other DP SQL systems.

System	DP policy		Query class		Utility
	Privacy level	Primary relation	Aggregation	Join	
PINQ [13]	Tuple-DP	/	Count	PK-FK Join	Low
FLEX [10]	Tuple-DP	/	Count	Unrestricted	Low
RS [5]	Tuple-DP	/	Count	Self-join free	Low
PrivateSQL [11]	User-DP	Single	Count	Self-join free	Low
ZetaSQL [14]	User-DP	Single	(Distinct)Count/Sum/Max/Quantile	Self-join free	Low
DOP-SQL (ours)	User-DP	Multiple	(Distinct)Count/Sum/Max/Quantile	Unrestricted	High

showing significant improvement in terms of utility, sometimes even by orders of magnitude. DOP-SQL has integrated these state-of-the-art mechanisms into one system, and automatically chooses the appropriate mechanism depending the query.

Extensibility. DOP-SQL is designed to be extensible. It can be integrated with any relational database supporting standard SQL, although we will be using PostgreSQL for our demonstration.

2 SYSTEM OVERVIEW

DOP-SQL consists of four components: (i) data is store in any relational database supporting standard SQL, which can either be local or hosted in the cloud; (ii) a query parser that checks the type of the input query, rewrites it appropriately, and then submits it to the SQL database for evaluation; (iii) a DP engine that takes the output of the rewritten query and produces privatized query answers using an appropriate down-neighborhood optimal DP mechanism; and (iv) a user interface that facilitates the interaction with the system, including setting privacy configurations. An illustration of our system can be viewed in Figure 1.

2.1 Privacy policy

A key concept in differential privacy is that of *neighboring instances*. While DP guarantees that the difference between any two neighboring cannot be confidently detected by the adversary, it is up to the application scenario to determine which instances are neighbors. In a single table where each individual user contributes just one tuple, the natural definition is that two instances that differ by one tuple are neighboring. This DP policy is known as *tuple-DP*, as the adversary cannot tell whether any particular tuple exists in the instance. This policy can be extended to the multi-relational model easily, i.e., two database instances are neighbors if they differ by one tuple in one of the relations. Indeed, all the early DP SQL systems adopted this DP policy. However, as pointed out in [11], this simple DP policy does not provide enough protection for relational databases since the relational model aims at capturing complicated relationships among different entities, so tuples in different relations shall not be treated equally. Consider the TPC-H database for instance, which consists of relations `Customer`, `Order`, and `Lineitems`, among others. Under tuple-DP, two neighboring instance may only differ by, say, one lineitem. If we remove one customer's data, which includes not only the corresponding tuple in `Customers`, but also all the associated tuples in `Orders` and `Lineitems`, then this would result in two instances that are not neighbors under tuple-DP, i.e., the customer's privacy is not protected. To accommodate this higher level of privacy protection, the *user-DP* policy has been formulated [11], which has been adopted by most subsequent DP SQL engines. Under user-DP, one or more relations are designated as *primary private relations*, which correspond to the "users" whose privacy we wish to protect, such as `Customer`. Another relation is called `secondary private relation` if it has a foreign-key referencing, directly or indirectly, a primary private relation, such as `Order` and `Lineitems`. Two instances are neighbors if one can be obtained from the other by removing one tuple t from a primary private relation and all the tuples in the secondary private relations that reference (directly or indirectly) t. Note that the user-DP policy is flexible enough to incorporate two commonly used DP policies over graph data. We can store a graph using the schema `Node(ID)`, `Edge(src, dst)`. Then by specifying `Node` as the primary private relation, user-DP degenerates into the *node-DP* policy; specifying `Edge` as the primary private relation corresponds to the *edge-DP* policy.

Additionally, when the database contains different types of "users", whose privacy should all be protected, multiple primary private relations shall be designated. Again consider the TPC-H data. If the privacy of the customers and suppliers should both be protected, then one may set both `Customer` and `Supplier` as primary private relations. Note that, however, additionally setting `Order` as a primary private relation would not be necessary, since the protection level brought by `Order` is subsumed by that of `Customer`. In general, a secondary private relation's privacy is automatically protected by the primary private relation its foreign key refers to. While the previous DP SQL systems only support a single primary private relation, DOP-SQL supports any number of relations to be designated as primary private relations.

2.2 Query syntax

DOP-SQL supports queries conforming to the following syntax:

```
SELECT < aggregation function >
 FROM {tables, ...}
{ JOIN < tables > on < bool expression > }
[ WHERE { bool expression, ...} ]
[ GROUP BY { tables.field / expression, ... } ]
```

Explanations on the notation:

- Square brackets [] indicate optional expressions with 0 or 1 appearance.
- Curly brackets {} indicate required expressions with 1 or multiple appearances.

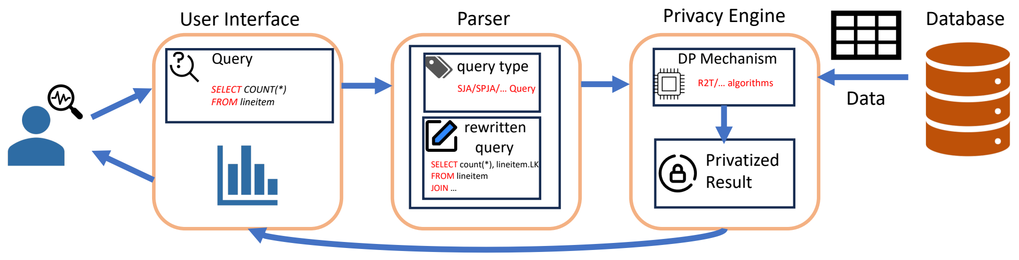


Figure 1: An overview of our system architecture with examples.

- Angle brackets <> indicate required expression with exact one appearance.

The aggregation function can be one of the following:

(1) `count(*)` or `count(DISTINCT *)`: This counts the number of (distinct) records in the results.
(2) `sum(expression)`: This returns the sum of an expression of some numerical columns.
(3) `max(expression, k)`: This returns the k-th largest value of an expression. Note that by setting an appropriate value for k, this function can be used to obtain the max, min, or any quantile.

All expressions follow the same syntax as standard SQL. DOP-SQL accepts both implicit joins (i.e., state the join conditions in the `WHERE` clause) or explicit joins (i.e., use `JOIN` keyword). As in standard SQL, table aliases can be created using the `AS` keyword to support self-joins.

2.3 Parser

DOP-SQL uses a parser modified from that of PostgreSQL. It determines the query type based on the abstract syntax so that an appropriate DP mechanism can be invoked by the DP engine; see Section 2.4 for more details.

Another important job of the parser is to rewrite the query. Many queries, such as the one below, when evaluated alone do not return the relationship between the query results (before aggregation) and the users, which is needed by the query engine. For such queries, the parser will complete the query following the PK-FK relationships until the primary private relations have been added. For example, the following query

```
SELECT sum(l_quantity) FROM Lineitems
```

will be written into the following query (assuming `Customers` is the primary private relation:

```
SELECT sum(l_quantity) FROM Lineitems
JOIN Orders on Lineitems.OK = Orders.OK
JOIN Customers on Orders.CK = Customers.CK
```

The SQL parser will also unpack the aggregation and include the PKs of the private relations into a reporting query, which will also be needed by the privacy engine. Thus, the final rewritten query will be:

```
SELECT l_quantity, Customers.CK FROM Lineitems
JOIN Orders on Lineitems.OK = Orders.OK
JOIN Customers on Orders.CK = Customers.CK
```

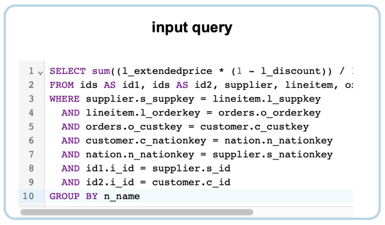
2.4 DP engine

The DP engine will automatically select an appropriate DP mechanism based on the query type. It takes the output of the rewritten query and produces a privatized answer. Currently, we have implemented the following mechanisms in the DP engine of DOP-SQL:

- For `COUNT` and `SUM` queries without self-joins, it uses the mechanism in [6].
- For `COUNT` and `SUM` queries with self-joins, it uses R2T [2].
- For `COUNT` and `SUM` queries with self-joins, the user may also choose an algorithm with a better optimality ratio (This algorithm will appear in the TODS version of [2]). It offers better utility but requires more time to execute.
- For `MAX/MIN/QUANTILE` queries without self-joins, it uses the exponential mechanism [12].
- For `MAX/MIN/QUANTILE` queries with self-joins, it uses the Shifted Inverse mechanism[9].
- For `COUNT DISTINCT` queries, it uses R2T.
- For `COUNT` and `SUM` queries with GROUP-BY, it uses the mechanism in [4] for answering multiple SJA queries.
- For other aggregation functions with GROUP-BY, it uses ShiftedInverse with basic composition (for pure DP) or advanced composition (for approximate DP).

2.5 User interface

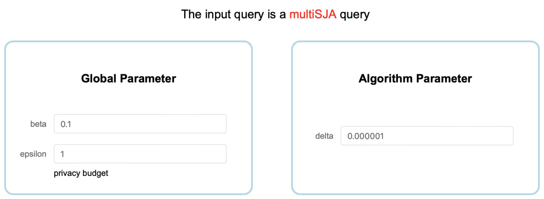
DOP-SQL provides a web-based interface for users to customize the database connection, and write SQL queries as shown in Figure 2a. The interface will also display all the relations available in the provided database where users can select private relations interactively. User can also set the configuration of the required parameters in DP mechanism for their desired privacy guarantees interactively as shown in 2b. Among all the required parameters in DOP-SQL, *epsilon* (privacy budget) is the most important one, which can determine privacy guarantee and accuracy. The impact of other parameters on privacy guarantee and accuracy is relatively minor. In addition, users can see the acyclic view of the database scheme where all the protected relation are colored. In this acyclic view, red nodes indicate primary private relations chosen by the users, and yellow nodes indicate secondary private relations which will also be protected in the results. Finally, user can see the privatized result from the query engine in a bar chart as shown in 2c.



input query

```sql
SELECT sum((l_extendedprice * (1 - l_discount)) /
FROM ids AS id1, ids AS id2, supplier, lineitem, o
WHERE supplier.s_suppkey = lineitem.l_suppkey
  AND lineitem.l_orderkey = orders.o_orderkey
  AND orders.o_custkey = customer.c_custkey
  AND customer.c_nationkey = nation.n_nationkey
  AND nation.n_nationkey = supplier.s_nationkey
  AND id1.i_id = supplier.s_id
  AND id2.i_id = customer.c_id
GROUP BY n_name
```

(a) Users can input a query.



The input query is a multiSJA query

Global Parameter

beta: 0.1

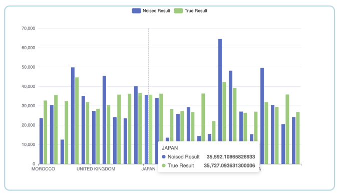
epsilon: 1

privacy budget

Algorithm Parameter

delta: 0.000001

(b) Users can customize privacy guarantee.



(c) The visualization of the privatized result.

Figure 2: Major interfaces of DOP-SQL.

3 DEMONSTRATION PLAN

During the demonstration, we will provide an interactive experience for audiences to get a better understanding of our system.

Step 1. The audience should first set up the database configuration. For ease of demonstration, we offer two local databases, one using the TPC-H benchmark and the other modeling a graph. Users can also choose to use their own schema by providing the server address and access account of the remote PostgreSQL database.

Step 2. After connecting to the database, audience need to submit a SQL query on the Query input component. For each of demonstration, we have prepared a large collection of SQL queries for the audience to choose from; they can of course write their own queries as well, following the syntax in the Section 2.2. Meanwhile, the audience can also set any subset of the relation in the schema as primary private relations that they want to protect in the database. If the user selects a pre-defined query, the primary private relations will be selected automatically using the default one. The SQL query will then be sent to the server.

Step 3. The query type and required parameter will be displayed in the interface. The audience can adjust the required parameters for the selected DP mechanism to meet their desired privacy guarantee. The audience can check the annotations next to the input box of parameters for simple explanation.

Step 4. After the query has been evaluated, the audience will get both the privatized result and the true results displayed in the Result display component using a bar chart. The audience can check the error level between privatized results and the true results in this chart. Detailed numerical result can be shown by moving cursor inside the bar chart. The processing time will also be displayed.

4 FUTURE WORK

As future work, we plan to extend the query class supported by DOP-SQL, such as sub-queries and recursion. Another interesting direction is to answer queries under updates. This is known as differentially private continue observation [8]. Earlier work has only studied the `COUNT` function over a single table. Recently, some progress has been made on answering SQL queries in this setting [1, 3], and we plan to include these mechanisms into DOP-SQL.

ACKNOWLEDGMENTS

This work has been supported by HKRGC under projects 16205420, 16205422, and 16204223.

REFERENCES

[1] Wei Dong, Zijun Chen, Qiyao Luo, Elaine Shi, and Ke Yi. 2024. Continual Observation of Joins under Differential Privacy. *Proc. ACM Manag. Data* 2, 3, Article 128 (may 2024), 27 pages. https://doi.org/10.1145/3654931

[2] Wei Dong, Juanru Fang, Ke Yi, Yuchao Tao, and Ashwin Machanavajjhala. 2022. R2T: Instance-optimal Truncation for Differentially Private Query Evaluation with Foreign Keys. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. ACM, New York, NY, USA, 759–772. https://doi.org/10.1145/3514221.3517844

[3] Wei Dong, Qiyao Luo, and Ke Yi. 2023. Continual Observation under User-level Differential Privacy. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 2190–2207. https://doi.org/10.1109/SP46215.2023.10179466

[4] Wei Dong, Dajun Sun, and Ke Yi. 2023. Better than Composition: How to Answer Multiple Relational Queries under Differential Privacy. *Proc. ACM Manag. Data* 1, 2, Article 123 (jun 2023), 26 pages. https://doi.org/10.1145/3589268

[5] Wei Dong and Ke Yi. 2021. Residual Sensitivity for Differentially Private Multi-Way Joins. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. ACM, New York, NY, USA, 432–444. https://doi.org/10.1145/3448016.3452813

[6] Wei Dong and Ke Yi. 2023. Universal Private Estimators. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '23)*. ACM, New York, NY, USA, 195–206. https://doi.org/10.1145/3584372.3588669

[7] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Theory of Cryptography*, Shai Halevi and Tal Rabin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 265–284.

[8] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. 2010. Differential privacy under continual observation. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing* (Cambridge, Massachusetts, USA) *(STOC '10)*. ACM, New York, NY, USA, 715–724. https://doi.org/10.1145/1806689.1806787

[9] Juanru Fang, Wei Dong, and Ke Yi. 2022. Shifted Inverse: A General Mechanism for Monotonic Functions under User Differential Privacy. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. ACM, New York, NY, USA, 1009–1022. https://doi.org/10.1145/3548606.3560567

[10] Noah Johnson, Joseph P. Near, and Dawn Song. 2018. Towards practical differential privacy for SQL queries. *Proc. VLDB Endow.* 11, 5 (oct 2018), 526–539. https://doi.org/10.1145/3177732.3177733

[11] Ios Kotsogiannis, Yuchao Tao, Xi He, Maryam Fanaeepour, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. 2019. PrivateSQL: a differentially private SQL query engine. *Proc. VLDB Endow.* 12, 11 (jul 2019), 1371–1384. https://doi.org/10.14778/3342263.3342274

[12] Frank McSherry and Kunal Talwar. 2007. Mechanism Design via Differential Privacy. In *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*. 94–103. https://doi.org/10.1109/FOCS.2007.66

[13] Frank D. McSherry. 2009. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data* (Providence, Rhode Island, USA) *(SIGMOD '09)*. ACM, New York, NY, USA, 19–30. https://doi.org/10.1145/1559845.1559850

[14] Royce Wilson, Celia Zhang, William Lam, Damien Desfontaines, Daniel Simmons-Marengo, and Bryant Gipson. 2020. Differentially Private SQL with Bounded User Contribution. *Proceedings on Privacy Enhancing Technologies* 2020 (04 2020), 230–250. https://doi.org/10.2478/popets-2020-0025