Interactive Preference-Aware Query Optimization

N. R. Ong #1, S. E. Rojcewicz #2, N. L. Farnan #3, A. J. Lee #4, P. K. Chrysanthis #5, T. Yu *‡6

Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, USA
{ 1nro5, 2srojcewicz, 3nlf4, 4adamlee, 5panos }@cs.pitt.edu

* Department of Computer Science, North Carolina State University, Raleigh, NC, USA
‡ Qatar Computing Research Institute, Doha, Qatar
6yu@csc.ncsu.edu

***Abstract*—PASQL is an extension to SQL that allows users of a distributed database to specify privacy constraints on an SQL query evaluation plan. However, privacy constraints can be difficult for users to specify, and worse yet, all possible situations that could lead to a privacy violation may not be known to the user a priori. To address these challenges, we propose a GUI-based interactive process for detecting such violations and generating appropriate constraints. In this work, we demonstrate two approaches to implementing such a GUI that provide different ways of analyzing and interactively optimizing a PASQL query plan.**

I. Introduction

The declarative nature of SQL has been a major strength of relational database systems: users simply specify what data they wish to access, and the Database Management System (DBMS) determines the best plan for accessing that data. Traditionally, the best plan is assumed to be the one with the shortest execution time, but that may result in a number of privacy violations.

As an example, assume a query plan joins `TableA` with `TableB` at server `SiteA`. If the query is executed, then `SiteA` becomes aware of the users' interest in data from both `TableA` and `TableB`. The fact that these two tables are being used *together* could be considered sensitive information, and a plan that reveals this information to an untrustworthy server may violate users' privacy. A standard DBMS does not reveal the query plan to users, and may unknowingly execute a plan that violates their desire to keep some aspects of the query hidden from some servers in the network.

To assuage these types of problems, in previous work we modified the PostgreSQL DBMS to implement a Preference-Aware Query Optimizer (PAQO) [2] that was proven to uphold declarative, user-specified constraints on query plan execution during the query optimization process. Users specify these constraints through an extension to SQL that we have developed called Preference-Aware SQL (PASQL) [3].

PAQO is a powerful tool, but two problems limit its effectiveness when it is used by itself. First, even while we assume users have a working knowledge of SQL, it is difficult for users to recognize the many ways in which a query plan could violate their privacy preferences, especially in advance. Second, PASQL syntax is complex and non-intuitive. These led us to augment PAQO with a GUI to allow users to explore a query plan and develop constraints capturing their privacy preferences. In this way, users can interactively define constraints and invoke PAQO to re-optimize the query plan until they are confident that the produced query plan satisfies their privacy concerns. As opposed to existing work (e.g., [5], [4], [1]) on interactive specification and refinement of SQL queries, our GUIs allow users to view and generate the query execution plan rather than the result of the query.

In this demonstration, we will show two GUI solutions that improve PAQO's usability through interactive query optimization while continuing to optimize runtimes. The first, referred to as the *Query View GUI*, is centered around the query specification whereas the second, the *Hierarchical View GUI*, is centered around the elements of constraints, i.e., data and execution sites. We demonstrate both of these approaches to interactive query optimization in order to gather feedback on their use to drive a future user study comparing the two.

II. Background

A. System Model and Scenario

In our work, we consider the system model depicted in Figure 1. The first step of the distributed query evaluation process is to determine the best plan for evaluating the query. This plan is divided into sub-plans, each of which is assigned to be evaluated by a particular server. Each server evaluates its portion of the query, and the final query result is returned to the querier. In this traditional setting, users possess no control over the distribution of partial query plans among the database servers. Hence, the system is free to choose a query plan that may violate users' privacy, as discussed above. To highlight this problem, consider the following scenario:

Example Scenario

Alice is a low ranking corporate executive who wishes to investigate possible illegal pollution by her employer, ManuCo. As a first step in this investigation, she wishes to join data stored by her employer with that of an environmental watchdog group (Pollution Watch) and a waterway mapping service (Mapper) to see if there is any correlation between where her company has plants holding industrial solvents and where those chemicals are appearing as waterway pollutants. To do so, she constructs a query over records describing hazardous solvents owned by ManuCo (stored in the `Supplies` table on ManuCo's `Inventory` database server), details of

978-1-4799-7964-6/15/$31.00 © 2015 IEEE

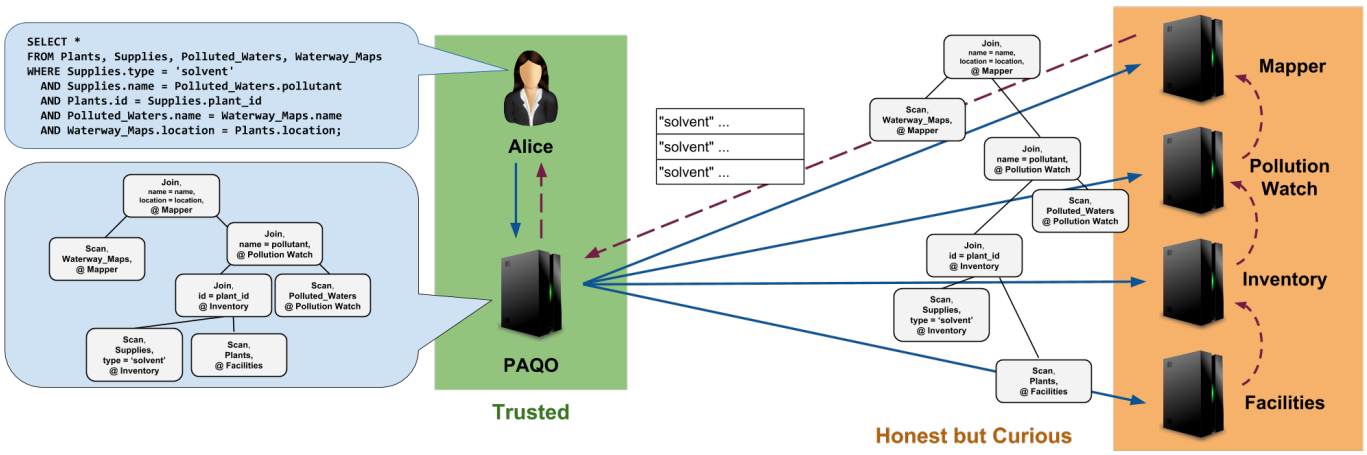


Fig. 1: A visualization of how PAQO would process Alice's example query from Section 2. Solid lines indicate Alice passing her query to PAQO, and PAQO distributing the partial evaluation plans to each server involved in the query. The final result of the query is passed back to Alice along the dashed line.

ManuCo's manufacturing plants (stored in the `Plants` table on ManuCo's `Facilities` server), water pollution data (stored in Pollution Watch's `Polluted_Waters` table), and waterway location data (stored in Mapper's `Waterway_Maps` table) as follows:

```
SELECT *
FROM Plants, Supplies, Polluted_Waters, Waterway_Maps
WHERE Supplies.type = 'solvent'
    AND Polluted_Waters.pollutant = Supplies.name
    AND Plants.id = Supplies.plant_id
    AND Polluted_Waters.name = Waterway_Maps.name
    AND Waterway_Maps.location = Plants.location;
```

In issuing this query, however, Alice would not want either ManuCo or Pollution Watch to become aware of the portion of her query that was issued to the other, or the join condition between the `Supplies` and `Polluted_Waters` tables. Such a revelation could easily cost her job, either because her employer felt that she "knew too much," or because the watchdog group applied external pressure to the company after learning of the intent of her query.

B. PASQL

To help Alice, we propose PASQL, which empowers users to impose constraints over the plans that evaluate their queries. PASQL constraints take the following form:

`REQUIRING` *condition 1* `HOLDS OVER` *node descriptor 1*
[`AND` *condition 2* `HOLDS OVER` *node descriptor 2*]

where a node descriptor is a ternary of the form:
$\langle relational\ operator, \{parameters\}, site \rangle$.
An explanation of the syntax and semantics of PASQL can be found in [3].

Consider our example scenario. Alice could use the following constraint to prevent the attribute `Supplies.name` from being evaluated at the server `Pollution_Watch`.

```
REQUIRING @p <> Pollution_Watch HOLDS OVER
    < *, {(Supplies.name)}, @p >
```

If Alice also wishes to force the JOIN on `Polluted_Waters` and `Supplies.name` to occur at either her local `Querier` server or at another site that she trusts, called `TrustedSite`, she would add the following constraint to her query.

```
REQUIRING @p IN {Querier, TrustedSite} HOLDS OVER
  < join, {(Polluted_Waters.pollutant,=, Supplies.name)}, @p >
```

Alice may encounter difficulty when attempting to produce the correct syntax. In addition, it may be difficult for Alice to determine the appropriate constraints to protect her privacy. We believe our GUIs mitigate both of these problems.

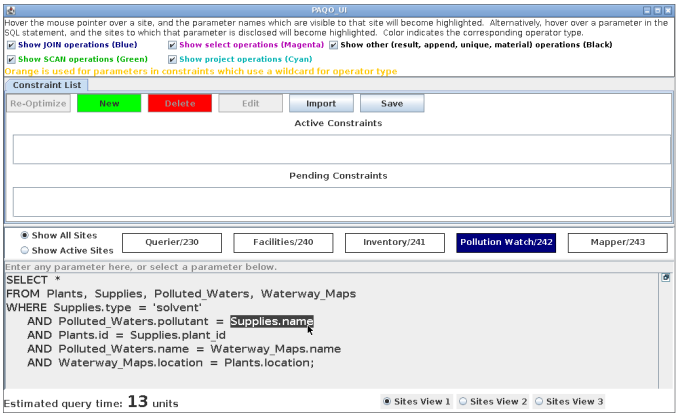
III. Graphical User Interface Approaches

We have developed two GUIs, namely, *Query View GUI* and *Hierarchical View GUI*, each of which is based on a different visual paradigm for analyzing query plans. Both provide users with a visual mapping of relationships within a query plan, i.e., how data moves throughout the distributed system's servers and where data is operated on.

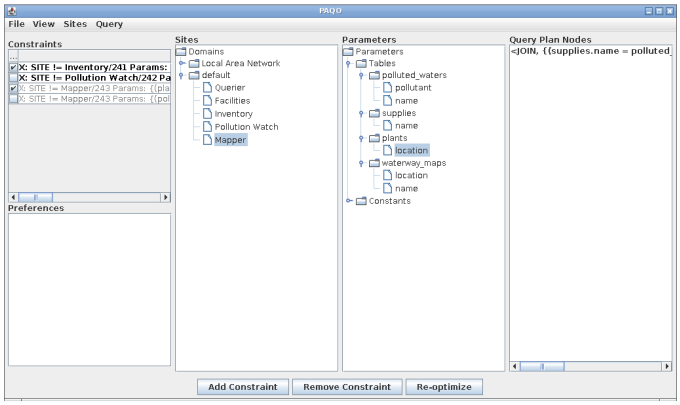
The GUIs provide two services: (1) present an interactive representation of the query execution plan, allowing users to explore the plan to aid them in the detection of privacy violations; and, (2) enable users to create constraints without requiring knowledge of PASQL syntax.

The workflow proceeds as follows:

1) Users specify an initial SQL query without any constraints. An initial query execution plan is generated.
2) With the aid of the GUI, users explore the query execution plan through the visual and interactive representation and look for violations of their privacy.
3) If users are not satisfied with the query plan, then they create appropriate constraints with GUI assistance, and generate a new query plan.
4) Steps 2-3 are repeated until users deem the query plan to be satisfactory.

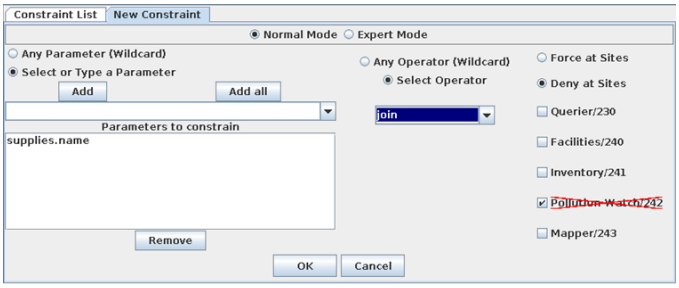


(a) Query View

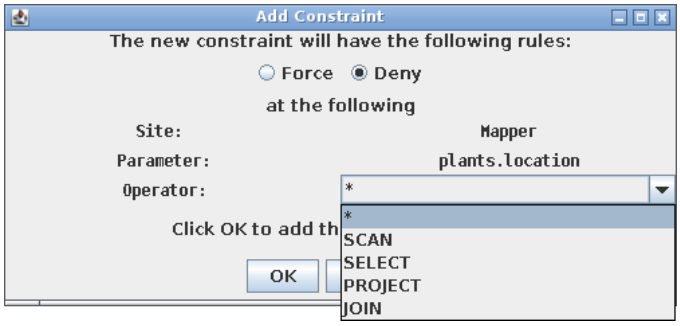


(b) Hierarchical View

Fig. 2: The main screens of each GUI.



(a) Query View



(b) Hierarchical View

Fig. 3: The Constraint Helper Window

A. *Query View GUI*

The Query View GUI is structured around the SQL query which is centrally displayed as shown in Figure 2a. It presents how data moves among servers and where data is operated on through mouse-hovering events. When the cursor is placed over a site, the parameters of the operations which occur at that site become highlighted in the SQL query. Similarly, when the cursor is placed on a parameter in the SQL query, it causes the sites which operate on that parameter to become highlighted. The color depends on the type of operation that is done to the selected data or at the selected site. In cases where multiple kinds of operations occur on the same data at the same site, multi-colored highlighting is displayed. A view can be customized, focusing on specific operations by appropriately selecting the colors in the top options panel. For example, unchecking the "Blue" option will disable highlighting for JOIN operations.

The constraint panel lies beneath the options panel, and is currently showing the listing of constraints. The panel contains two lists of constraints, one for constraints already included in the current query execution plan (Active Constraints), and one for constraints that have yet to be included in the query plan (Pending Constraints). Newly added constraints are placed in the Pending Constraints listing, and moved to the Active Constraints list upon re-optimization.

A panel of sites is presented beneath the constraint panel. Users are able to see the full listing of sites, and can restrict the list to only sites that are used in the query plan. If users hover over a site that processes some data in the current query plan then that data will be highlighted with the corresponding colors in the SQL query below. Users may also hover over elements in the SQL query, and sites that handle that data will be highlighted with the SQL operation color.

Using this view, users can easily determine site and data relationships visually in clear color, which will be useful for users analyzing the query plan for privacy violations. In Figure 2a, the mouse is hovering over the `Supplies.name` attribute, so the `Pollution_Watch` site becomes highlighted. This indicates that `Pollution_Watch` operates on `Supplies.name`. The blue color of the highlighting indicates that the operation is a JOIN. As given in our example scenario, this query plan violates Alice's privacy requirement.

Figure 3a shows the constraint creation interface as it is used to create Alice's constraint. The left side of the window shows a list of parameters to constrain. In the center is the selected operation, with the possibility of selecting the wildcard. If more than one operation was detected in the data/site pairing, then the system defaults to the wildcard operation. The right

side provides a full listing of sites, where the checkboxes indicate at which sites data can or cannot be operated at, depending on the "Force" or "Deny" options. Clicking the "OK" button finishes the constraint creation, placing the user's newly created constraint in the Pending Constraints list.

Once users are finished creating new constraints, they can click the "Re-Optimize" button to generate a new query plan. Once optimization is finished, the GUI displays to the users a comparison between the new query plan and original, with the option to revert to the previous query plan.

B. Hierarchical View GUI

The Hierarchical View GUI allows users to focus on the elements of constraints that are most prominent in protecting their privacy, namely sites and data parameters. Users are given the query plan in a site- or data-oriented view, where a focus on a specific site exposes all data parameters that it comes in contact with, and vice versa. Once users select sites and data parameters, a list of applicable query plan nodes is displayed to enable the user to detect privacy violations in the query plan. Users may then create constraints that correspond to the sites, parameters, and optionally nodes, that they have selected. These new constraints are then included in a list of user-defined constraints, which can be selected for re-optimization.

Users are given a choice on which aspect of the query plan to use as a starting point for exploration. In the *Site View*, a full list of sites is presented in a Domain-specified hierarchical structure. Selecting a site presents all data parameters associated with that site. Conversely, in the *Data View*, a full list of data parameters is presented first, divided by tables and constant types. Tables can be further subdivided into their corresponding attributes that are relevant to the SQL query.

Figure 2b displays the main view of this GUI (shown in Site View). On the far left, the lists of user-created constraints are provided, one for requirements and one for preferences. Users can specify which constraints are to be included in the next optimization cycle by using the check boxes. Constraints that are already included in this optimization cycle are in bold, while those that are not are gray-colored.

The next panel to the right of the constraints panel is dependent on the view. If Site View is used, then the Sites Panel will provide the full list of sites at that position, and the Parameter Panel will be to the right. If the Data View is used, then the Parameter Panel will be shown, with the Sites Panel to the right. The rightmost panel is the Node Panel, which lists all query plan nodes that contain the selected site(s) and parameter(s).

Once the user is satisfied with exploring, she also has the option of selecting elements that she feels violate her privacy. By choosing sites, parameters, and eventually query plan nodes, users can add constraints based on their selections. However, a constraint need-not be defined by a selection in the Sites, Parameter, and Node Panels. A lack of a selection in one will trigger a drop down menu in the Constraint Helper Window (Figure 3b). In addition, the wildcard can also be selected in this instance.

These constraints will then be added to the Constraint Panel on the left and ready for re-optimization.

IV. Demonstration

In this demonstration, we first illustrate how our GUI tools help to inform users about where the component pieces of their query will be disseminated during evaluation and hence help them to detect violations of their privacy before they occur. From here, we show how our tools can further ease users through constraining the plans that can be generated to evaluate their queries so that such privacy violations are avoided. Finally, both GUIs will also inform users of the runtimes of their new constrained query execution plans (if they exist). Specifically, we utilize the example scenario and query from Section II, and demonstrate the process of obtaining a query plan that satisfies Alice's privacy needs.

Our demonstration begins with the generation of a plan for Alice's query without any specified constraints. We show how both GUIs present the query execution plan to users in a way that highlights where portions of their queries are being evaluated. We show how users can explore these interactive representations of their queries to identify privacy violations. We then illustrate how our GUI tools aid users in creating constraints that can protect their privacy by populating constraints in response to the state of the query plan exploration. We then re-optimize the now constrained query through our GUI tools and show how users can explore the new query execution plan. Users can ensure their constraints have the desired effect on query evaluation and compare the estimated costs of the two plans.

Since it is possible to use different sets of constraints to satisfy the privacy requirements in our example scenario, we further demonstrate the features of our GUIs by repeating the interactive and iterative optimization process using different constraints. We also discuss the minimal impact that upholding constraints has on query *optimization* time. Finally, we invite participants to experiment with our user interface, and provide feedback that will help us to develop a future user study comparing the two GUI tools.

Acknowledgments

This work was supported in part by NSF awards CCF–0916015, CNS–0964295, CNS–0914946, CNS–0747247, OIA–1028162, and CNS–1253204, and by a gift from EMC/Greenplum. We thank the reviewers for their feedback.

References

[1] C. Cerullo and M. Porta. A system for database visual querying and query visualization: Complementing text and graphics to increase expressiveness. In *DEXA Workshops*, 2007.

[2] N. L. Farnan, A. Lee, P. Chrysanthis, and T. Yu. PAQO: Preference-aware query optimization for decentralized database systems. In *ICDE*, 2014.

[3] N. L. Farnan, A. J. Lee, P. K. Chrysanthis, and T. Yu. Don't reveal my intension: Protecting user privacy using declarative preferences during distributed query processing. In *ESORICS*, 2011.

[4] D. A. Keim and V. Y. Lum. Visual query specification in a multimedia database system. In *IEEE Visualization*, 1992.

[5] C. Mishra and N. Koudas. Interactive query refinement. In *EDBT*, 2009.