QO-Insight: Inspecting Steered Query Optimizers

Christoph Anneser, TUM, anneser@in.tum.de
Mario Petruccelli, TUM, petruccelli@in.tum.de
Nesime Tatbul, Intel Labs and MIT, tatbul@csail.mit.edu
David Cohen, Intel, dave.cohn@gmail.com
Zhenggang Xu, Meta, zhenggang@fb.com
Prithviraj Pandian, Meta, prithvip@fb.com
Nikolay Laptev, Meta, nlaptev@fb.com
Ryan Marcus, Univ. of Pennsylvania, ryan@ryanmarc.us
Alfons Kemper, TUM, kemper@in.tum.de

ABSTRACT

Steered query optimizers address the planning mistakes of traditional query optimizers by providing them with hints on a per-query basis, thereby guiding them in the right direction. This paper introduces QO-Insight, a visual tool designed for exploring query execution traces of such steered query optimizers. Although steered query optimizers are typically perceived as black boxes, QO-Insight empowers database administrators and experts to gain qualitative insights and enhance their performance through visual inspection and analysis.

PVLDB Reference Format:
Christoph Anneser, Mario Petruccelli, Nesime Tatbul, David Cohen, Zhenggang Xu, Prithviraj Pandian, Nikolay Laptev, Ryan Marcus, and Alfons Kemper. QO-Insight: Inspecting Steered Query Optimizers. PVLDB, 16(12): 3922 - 3925, 2023.
doi:10.14778/3611540.3611586

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/christophanneser/QO-Insight.

1 INTRODUCTION

Query optimizers are highly complex software systems. Typically, large engineering teams develop them to serve various workloads under different conditions. As a result, optimizers employ multiple heuristics and tuning knobs coupled with cost models based on statistics and estimates, which increases the risk of query planning mistakes that may lead to slower queries. To overcome this long-standing problem, researchers have recently turned to learning-based solutions [14].

The Bandit optimizer (Bao) provides an ML-based, practical enhancement to a traditional query optimizer [6]. Given a collection of *hint-sets*, where each hint-set defines which subset of query rewrite rules should be considered in query planning, Bao learns to *steer* a traditional query optimizer by choosing the right hint-set for every incoming query. This way, many planning mistakes of traditional query optimizers can be avoided. Since the first Bao-for-PostgreSQL prototype, Bao has been applied to several commercial and open-source optimizers, including its production use at Microsoft [13]. In our recent work, we have further extended Bao with a novel hint-set discovery approach, generalizing it into an open-source steering framework called AutoSteer, which is applicable to many SQL databases [1].

While steered query optimizers have been shown to outperform traditional query optimizers in various benchmarking studies [1, 6, 8, 13], there has been little work into *why* query plans generated by learned query optimizers outperform those generated by traditional optimizers. In other words, detailed qualitative evaluations are needed to better analyze traditional query optimizers compared to their steered counterparts. This observation motivated us to build QO-Insight – the visual exploration tool we propose in this paper.

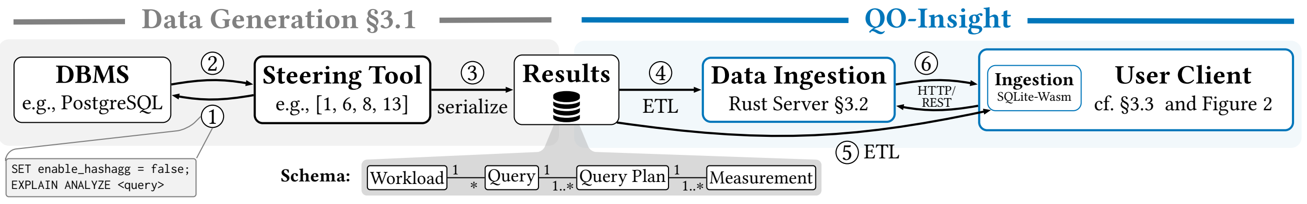
QO-Insight accepts query execution traces from a steered query optimizer as input and provides a visual front end to interactively analyze these traces to gain insights. Key design features of our tool include (i) multiple exploration modes (query-centric vs. rule-centric), (ii) support for multiple performance metrics and their arbitrary combinations (e.g., latency, number of page spills), (iii) visual comparison of query plans along with their important meta-data, and (iv) support for interactive drill-downs and aggregations to efficiently retrieve the relevant information.

We will demonstrate QO-Insight and its usefulness through two demonstration scenarios. The first one shows how a database administrator (DBA) can use QO-Insight's query-centric exploration mode to tune workloads according to their custom requirements. The second one shows how a query optimization expert (QOE) can leverage QO-Insight's rule-centric exploration mode to find problematic patterns and weaknesses in the underlying query optimizer. Through the presentation of these example usage scenarios, we hope to expose our audience to the technology behind steered query optimizers and their impact on improving query performance.

2 RELATED WORK

Both query optimizers and visualization tools to interactively analyze them through the plans they generate have been popular demonstration topics at past database conferences. A handful of these focus on debugging queries for logical correctness (e.g., Habitat [3] for recursive queries and I-Rex [7] for provenance-based comparative testing), while most – like QO-Insight – focus on understanding issues related to query performance. For example, Picasso [4] provides a suite of diagrams that qualitatively and quantitatively describe the plan choices made by an optimizer; QE3D [11] enables three-dimensional visualization for analyzing the performance of distributed query plans; Candomble [9] is for interactively examining plans for hardware-conscious query optimization. There

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611586



```
SET enable_hashagg = false;
EXPLAIN ANALYZE <query>
```

Figure 1: One of the recently proposed steering approaches connects to a SQL database, uses its exposed knobs to steer query optimization ①, and serializes query plans and performance results ③. QO-Insight loads and transforms the data either on a dedicated server ④ or on the client's browser ⑤, and provides it to the user client which allows for interactive exploration ⑥.

are also tools built specifically to support database education. For example, DBInsight [10] visualizes query processing pipelines end to end; MOCHA [12] explores the impact of alternative physical operator choices on the query execution plan of a given SQL query. Closer to our work, Phints [2] provides a framework for visually specifying query hints in Microsoft's SQL Server so that DBAs can debug poorly performing queries and identify and force hints that would lead to better plans.

While generally similar in spirit to all of these previous visual query optimizer tools, QO-Insight's key novelty lies in its focus on the newly emerging class of *steered query optimizers* [1, 6, 8, 13]. These optimizers integrate an ML-based adaptive feedback loop into traditional query optimizers while leveraging their query hinting mechanisms by selectively turning off optimizer rules that may lead to planning mistakes. This process generates much training data with rich information on the underlying database and its optimizer's behavioral patterns across various workloads and performance metrics. QO-Insight provides a convenient interface for users to visually explore these patterns – not only to understand when and why adaptive steering is helpful but also to discover ways to improve the native optimizer itself to deliver better query performance.

3 SYSTEM OVERVIEW

This section describes QO-Insight's data generation, the data ingestion, the user interface, and the query plan matching algorithm.

3.1 Data Generation

As shown in Figure 1, QO-Insight works with many of the recently proposed approaches in [1, 6, 8, 13] that leverage the databases' exposed knobs to steer query optimization ①. Next, the explored query plans and performance measurements ② are exported and serialized, e.g., to a JSON file or an SQLite database ③. The evaluation results are structured as follows: *workloads* contain an arbitrary number of *queries*, for which the steering approach generates one or more *query plans* and collects at least one *measurement* for each plan. Based on the measurements, we evaluate the query plan performance by considering several metrics, such as the wall time, the number of page accesses, and the memory footprint. The proposed schema allows QO-Insight to extract meaningful results and to apply drill-downs and aggregations efficiently during data ingestion.

At https://github.com/christophanneser/QO-Insight, users can start exploring QO-Insight using three workloads collected with AutoSteer [1] for PostgreSQL: the Join Order Benchmark [5] (137 queries) and TPC-H with scale factors 1 and 10 (22 queries each).

3.2 ETL and Data Ingestion

Before users can explore the evaluation results in QO-Insight, we need to extract, transform, and load (ETL) the data into a format processable by the user client. We require efficient data processing to allow users to drill down and aggregate the results interactively, providing only the data the client needs. QO-Insight provides two options to perform the ETL steps: First, data processing can be performed on a dedicated backend server ④ to analyze large datasets, while datasets smaller than 4GB can be processed directly *inside* the browser ⑤. We implement an efficient, dedicated ETL server, which exposes the data via REST endpoints ⑥. The second option leverages SQLite-Wasm to preprocess small datasets entirely in the browser, offloading computation from the dedicated backend server to the client's device.

All endpoints allow the clients to apply filters to fetch only the required data. E.g., the serialized query plans attributed to 90% of the size of the result. However, QO-Insight does not need them for most of its functionality.

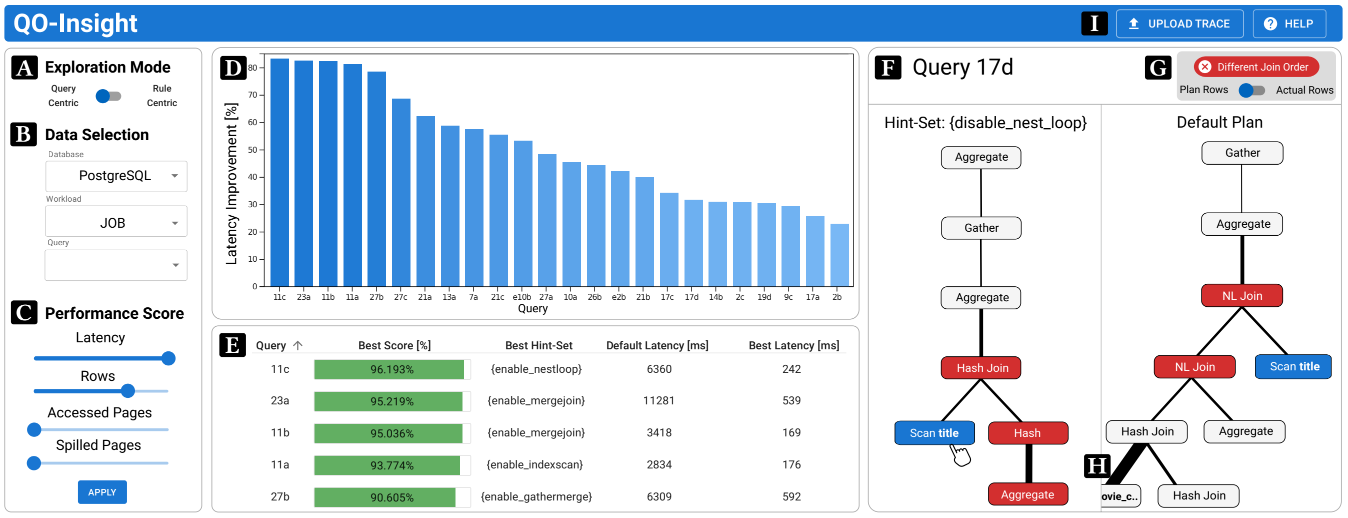
3.3 QO-Insight's User Interface

Figure 2 provides an overview of QO-Insight's user interface, which comprises multiple components that allow DBAs and QOEs to efficiently retrieve the information they are interested in. To the left side, QO-Insight has a **main menu** A - C that allows users to select *how* the data is visualized, *what* data they want to explore, and how the *performance scores* are calculated.

A **Exploration Mode.** Users can switch between a *query-centric* and a *rule-centric* exploration mode. In the *query-centric mode*, QO-Insight aggregates the evaluation results (e.g., the latency or the memory improvements) by queries or workloads. This mode supports DBAs and shows them the queries and workloads that can be improved the most. Contrary, the *rule-centric mode* aggregates the results according to hint-sets and shows QOEs those rules that had the most significant impact on performance.

B **Data Selection.** The data selection component allows users to define what data they want to explore. In the *query-centric* mode, three drop-down menus enable users to select a database instance and to drill down to workloads and queries, for which QO-Insight should present the results. Once the data selection changes, the results shown in D, E, and F will update accordingly. In the *rule-centric* mode, B looks slightly different than in Figure 2, allowing users to select *hint-sets*.

C **Performance Score.** QO-Insight supports *user-defined* performance scores to quantify and evaluate query plans. The overall



Query	Best Score [%]	Best Hint-Set	Default Latency [ms]	Best Latency [ms]
11c	96.193%	{enable_nestloop}	6360	242
23a	95.219%	{enable_mergejoin}	11281	539
11b	95.036%	{enable_mergejoin}	3418	169
11a	93.774%	{enable_indexscan}	2834	176
27b	90.605%	{enable_gathermerge}	6309	592

Figure 2: Screenshot of QO-Insight's user interface showing the query-centric mode when selecting a database and a workload.

score can be any combination of individual performance metrics, such as *latency*, *number of processed rows*, *number of accessed pages*, and *number of pages spilled to disk*, depending on what data the steering tool or the database makes available. Users can adjust the weight of each metric via sliders. When they apply new weights, QO-Insight compares each performance number to the default query plan, calculates the relative changes, and multiplies them with the user-defined weights, which results in the overall *score*, which will be higher for better plans, 0 for the default plan, and negative for worse plans.

Once QO-Insight has calculated the performance scores for the individual query plans, it *aggregates* the query and workload scores in the query-centric and the hint-sets scores in the rule-centric mode. QO-Insight defines the query score as the *highest score* achieved by one of the steered query plans and the workload score as the *weighted average* of all query scores belonging to that workload. To calculate the weighted average, QO-Insight sums the individual metrics of the best-scoring and the default query plans and, as before, multiplies the relative changes with the user-defined weights. QO-Insight calculates the performance scores within the user client without requiring any other interactions with the server.

D - E Displaying Performance Results. Depending on the settings in the main menu, QO-Insight visualizes the selected data in two ways: **D** leverages a bar chart to show the performance scores for workloads, queries, or query plans in the query-centric mode and the performance scores of hint-sets or their query plans in the rule-centric mode. The bars are sorted according to their performance score, allowing our users to focus on the queries having the most significant improvement potential. **E** provides more information, such as the number of accessed and spilled pages.

We designed all components for interactivity. For example, users can filter, sort, and export tables. Furthermore, they can click on bars or table rows to drill down into workloads, queries, hint-sets, and query plans, allowing for interactive exploration.

F Comparing Query Plans. When showing the results for a single query in the query-centric mode, QO-Insight's users can select two query plans by clicking on the bars, which opens **F** in full-screen mode and shows the two query plans side-by-side. We use the matching algorithm from Section 3.4 to identify common nodes and to highlight the differences between the two plans. When hovering over a matched node, both nodes are highlighted in blue, and a tooltip provides more details. Furthermore, users can switch between the *estimated* and the *actual* number of rows processed by each operator, configurable through the toggle in **G**.

I Upload Trace Files. QO-Insight provides two methods for data ingestion (cf. Section 3.2). For the analysis of large database traces, a dedicated backend server is employed. Alternatively, small trace files, with sizes up to 4 GB, can be seamlessly processed within the browser using SQLite Wasm. Users can submit their custom trace files by clicking on the upload button in the app bar **I**.

3.4 Comparing Query Plans with DiffPlan

Both DBAs and QOEs can use QO-Insight to explore steered query optimizers. As explained in Section 1, the steering approach uses hint-sets that turn the database's exposed knobs on and off to generate new, alternative query plans. This approach has demonstrated effectiveness across various systems, such as PostgreSQL [6], PrestoDB [1], and Microsoft SCOPE [8]. It improves query performance by up to 90% and helps QOEs identify weaknesses in existing optimizers by generating concrete *counter-examples* for which the optimizer performed poorly. However, comparing query plans becomes increasingly more complex the more tables are joined. Therefore, we implemented DiffPlan, a simple matching algorithm for query plans that finds common nodes and detects differences. It consists of three steps:

(1) **Top-Down:** First, it matches the root nodes and then traverses both trees downwards while identifying equal nodes.
(2) **Bottom-Up:** For the first query plan, we store all leaf nodes (e.g., table scans and materialized views) in a hash set $\mathcal{L}$. Then, we traverse the leaf nodes of the second plan and calculate their matching pipeline p for each matching node $l \in \mathcal{L}$. When

there are multiple matching leaf nodes in $\mathcal{L}$, we select those leaf nodes having the longest matching pipeline p in common.

(3) **Fuzzy Matching:** For each leaf node in the first plan, we traverse the tree bottom-up until we find a matched node n_1 with an unmatched parent, followed by another upwards traversal to the next matched node n_2. Next, we hash the unmatched nodes between n_1 and n_2 and search for potential matches in the path between nodes m_1 and m_2, with m_k being the node in the second plan matched to node n_k from the first plan.

Currently, we use a simple equality check, where inner nodes must have the same operator type and implementation details and leaf nodes must scan the table or the materialized view. We will extend DiffPlan to allow for custom equality criteria in the future.

We use the compiler *Emscripten* to translate the `C++` code to WebAssembly and run it *inside the browser* at near-native speed.

4 DEMONSTRATION SCENARIOS

This section demonstrates how users will interact with QO-Insight.

Database Admins (DBAs). Let us now walk through a typical scenario where a DBA aims to optimize frequently running workloads. The DBA previously used one of the approaches [1, 6, 8, 13] to steer the PostgreSQL optimizer for different workloads running on a cloud server. After loading the dataset directly into QO-Insight's user interface (not shown here), she uses **A** to switch to the *query-centric* exploration mode, which shows the potential improvements for each workload in a bar chart.

By clicking on the bar with the most significant improvements, the DBA drills down to an overview of the workload's queries and their potential gains and **B** updates accordingly. As the selected workload belongs to a dashboard comprising many widgets, the DBA is interested in the tail latencies, which she sorts in descending order in the table **E**. For example, the longest-running query takes over 11 seconds to execute, but the steering approach found an alternative plan that reduces the latency to 0.5 seconds.

By clicking on the query row in the table, the DBA drills down again, and QO-Insight now shows all alternative query plans. However, as the database instance runs in the cloud, the DBA must consider other metrics, such as memory consumption and network transfers, to reduce the operating costs. Therefore, she changes the performance metric in **C** to include the number of processed rows, accessed, and spilled pages. Then, she selects the hint-set yielding the best score and applies it to the production workload. E.g., she could instruct the DBMS to apply the hint-set automatically.

Query Optimization Experts (QOEs). Let us now walk through a second scenario in which a QOE wants to understand how she can improve the design and the implementation of the database's optimizer. As in the previous scenario, she first loads the data into QO-Insight. This time, however, the QOE switches to the *rule-centric* exploration mode in **A**, which aggregates the evaluation results *by hint-sets* rather than by workloads or queries. Component **D** now shows a bar chart with the potential improvements for each hint-set, sorted in descending order, and supporting her in identifying the hint-sets with the most potential for improvements.

Once she clicks on one of the bars, QO-Insight drills down again and shows a list of queries now, for which the hint-set improved the performance score compared to the query's default plan. Each of these plans is a counter-example that shows that better plans exist that the database's optimizer could not find.

When she clicks on one of the bars now, QO-Insight shows the default and the steered query plans side-by-side in **F**. Based on the matching algorithm described in Section 3.4, nodes that are different between the two plans are visually highlighted in red. Now, she can interactively explore both plans and, e.g., switch between the estimated and the actual cardinalities **G**, indicated by line widths.

As she investigates an increasing number of query plans with QO-Insight, she identifies frequently occurring and problematic patterns in PostgreSQL: (1) The larger tables are frequently chosen for the *build* sides of hash joins (e.g., **H**) despite the optimizer's awareness of their cardinality. (2) Index scans negatively impact the execution time frequently due to underestimated selectivities.

Overall, QO-Insight provides a user-friendly interface for qualitatively exploring steered query optimizers, supporting DBAs and QOEs in identifying performance bottlenecks.

5 CONCLUSIONS

This paper introduced QO-Insight, a visualization tool allowing users to inspect steered query optimizers. Our usage scenarios have demonstrated how QO-Insight supports DBAs in tuning their database workloads and QOEs in improving the query optimizer.

REFERENCES

[1] Christoph Anneser, Nesime Tatbul, David Cohen, Zhenggang Xu, Prithviraj Pandian, Nikolay Laptev, and Ryan Marcus. 2023. AutoSteer: Learned Query Optimization for Any SQL Database. *PVLDB* 16, 12 (2023), 3515–3527.

[2] Nicolas Bruno, Surajit Chaudhuri, and Ravishankar Ramamurthy. 2009. Interactive Plan Hints for Query Optimization. In *SIGMOD Conference*. ACM, 1043–1046.

[3] Benjamin Dietrich and Torsten Grust. 2015. A SQL Debugger Built from Spare Parts: Turning a SQL: 1999 Database System into Its Own Debugger. In *SIGMOD Conference*. ACM, 865–870.

[4] Jayant R. Haritsa. 2010. The Picasso Database Query Optimizer Visualizer. *PVLDB* 3, 2 (2010), 1517–1520.

[5] Viktor Leis, Bernhard Radke, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. 2018. Query Optimization Through the Looking Glass, and What We Found Running the Join Order Benchmark. *VLDB J.* 27, 5 (2018), 643–668.

[6] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Making Learned Query Optimization Practical. In *SIGMOD Conference*. ACM, 1275–1288.

[7] Zhengjie Miao, Tiangang Chen, Alexander Bendeck, Kevin Day, Sudeepa Roy, and Jun Yang. 2020. I-Rex: An Interactive Relational Query Explainer for SQL. *PVLDB* 13, 12 (2020), 2997–3000.

[8] Parimarjan Negi, Matteo Interlandi, Ryan Marcus, Mohammad Alizadeh, Tim Kraska, Marc T. Friedman, and Alekh Jindal. 2021. Steering Query Optimizers: A Practical Take on Big Data Workloads. In *SIGMOD Conference*. ACM, 2557–2569.

[9] Holger Pirk, Oscar R. Moll, and Sam Madden. 2016. What Makes a Good Physical plan?: Experiencing Hardware-Conscious Query Optimization with Candomblé. In *SIGMOD Conference*. ACM, 2149–2152.

[10] Ying Rong, Hui Li, Kankan Zhao, Xiyue Gao, and Jiangtao Cui. 2022. DBinsight: A Tool for Interactively Understanding the Query Processing Pipeline in RDBMSs. In *CIKM*. ACM, 4960–4964.

[11] Daniel Scheibli, Christian Dinse, and Alexander Boehm. 2015. QE3D: Interactive Visualization and Exploration of Complex, Distributed Query Plans. In *SIGMOD Conference*. ACM, 877–881.

[12] Jess Tan, Desmond Yeoh, Rachael Neoh, Huey-Eng Chua, and Sourav S. Bhowmick. 2022. MOCHA: A Tool for Visualizing Impact of Operator Choices in Query Execution Plans for Database Education. *PVLDB* 15, 12 (2022), 3602–3605.

[13] Wangda Zhang, Matteo Interlandi, Paul Mineiro, Shi Qiao, Nasim Ghazanfari, Karlen Lie, Marc T. Friedman, Rafah Hosn, Hiren Patel, and Alekh Jindal. 2022. Deploying a Steered Query Optimizer in Production at Microsoft. In *SIGMOD Conference*. ACM, 2299–2311.

[14] Rong Zhu, Ziniu Wu, Chengliang Chai, Andreas Pfadler, Bolin Ding, Guoliang Li, and Jingren Zhou. 2022. Learned Query Optimizer: At the Forefront of AI-Driven Databases. In *EDBT*. 1–4.