MOCHA: A Tool for Visualizing Impact of Operator Choices in Query Execution Plans for Database Education

Jess Tan
Nanyang Technological University
Singapore
jess0057@e.ntu.edu.sg

Desmond Yeo
Nanyang Technological University
Singapore
yeok0047@e.ntu.edu.sg

Rachael Neoh
Nanyang Technological University
Singapore
rach0081@e.ntu.edu.sg

Huey-Eng Chua
Nanyang Technological University
Singapore
hechua@ntu.edu.sg

Sourav S Bhowmick
Nanyang Technological University
Singapore
assourav@ntu.edu.sg

ABSTRACT

The database systems course is offered in many major universities. A key learning goal of learners taking such a course is to understand how SQL queries are processed in an RDBMS in *practice*. To this end, comprehension of the impact of various physical operators on the selected query execution plan (QEP) of a query is paramount. Unfortunately, off-the-shelf RDBMS typically only expose the QEP to users without revealing information about the impact of alternative choices of various physical operators on it in a *user-friendly* manner to aid learning. In this demonstration, we present a novel system called MOCHA that facilitates exploration and visualization of the impact of alternative physical operator choices on the QEP of a given SQL query. MOCHA accepts an SQL query as input, and *compares* and *visualizes* the QEP and *alternative plans* which are selected based on learner-specified *operator preferences*. Furthermore, it intuitively *explains* why the key operators in a QEP are chosen by connecting them to established knowledge in the literature.

PVLDB Reference Format:
Jess Tan, Desmond Yeo, Rachael Neoh, Huey-Eng Chua, and Sourav S Bhowmick. MOCHA: A Tool for Visualizing Impact of Operator Choices in Query Execution Plans for Database Education. PVLDB, 15(12): 3602 - 3605, 2022.
doi:10.14778/3554821.3554854

1 INTRODUCTION

The growing demand for "lifelong learning" coupled with the widespread usage of relational database management systems (RDBMS) in the commercial world and the growth of Data Science as a discipline have generated increasing demand of database-related courses in academic institutions. Learners from diverse fields and experiences aspire to take these courses, even with limited Computer Science backgrounds [5]. One of the key goals for learners taking a database course is to understand the execution strategies of SQL queries by an RDBMS. Given an SQL query, the query engine in an RDBMS produces a *query execution plan* (QEP), which represents the execution strategy of the query. Hence, such an understanding can be gained by perusing the QEPs of corresponding queries.

Proceedings of the VLDB Endowment, Vol. 15, No. 12 ISSN 2150-8097.
doi:10.14778/3554821.3554854

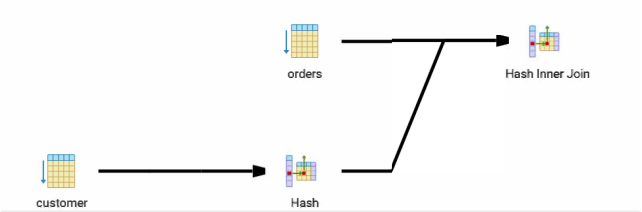

Figure 1: Visual tree representation of a QEP in PostgreSQL.

Major database textbooks (*e.g.,* [3]) introduce *general* (*i.e.,* not tied to any specific RDBMS) theories and principles associated with QEPs using natural language-based narratives. This allows a learner to gain a general understanding of query execution strategies. In particular, these textbooks typically illustrate QEPs of SQL queries, their estimated costs, and adverse impact on the cost if alternative physical operators are chosen (*e.g.,* merge join instead of hash join).

Most database courses complement text book-based learning with hands-on interaction with an off-the-shelf RDBMS (*e.g., PostgreSQL*). These RDBMSs are designed primarily for commercial purpose and not for pedagogical support. Consequently, in contrast to database textbooks, they only reveal the chosen QEP of a query in *visual* or *textual* format. Typically, they do not expose the impact of alternative choices of various physical operators on the QEP in a *user-friendly* manner to aid learning. Note that such information is invaluable to learners as it not only facilitates hands-on inquire-driven learning on the impact of a choice of a physical operator on the cost of a QEP but it also enables them to comprehend why a QEP is chosen by the underlying RDBMS. However, an RDBMS (*e.g., PostgreSQL*) typically demands a learner to manually pose an SQL query with various constraints on the *configuration parameters* to view the corresponding QEP containing specific physical operators. Furthermore, one has to manually compare the generated plan (referred to as *alternative query plan* (AQP)) with the original QEP to understand the impact. Notably, an undergrad database course typically does not introduce these configuration parameters while exposing syntax and semantics of SQL. Consider the following motivating scenario.

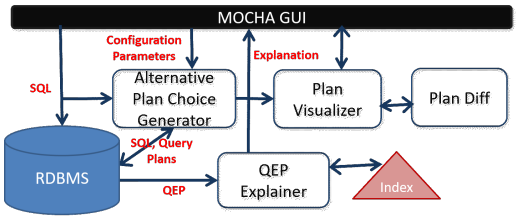

Figure 2: Architecture of MOCHA.

Example 1.1. Doreen is an undergraduate student who is currently enrolled in a database systems course. She wishes to understand the execution of an SQL query involving a join between the customer and orders relations on a TPC-H benchmark dataset in *PostgreSQL*. The corresponding visual representation of the QEP is depicted in Figure 1. Doreen wonders what will be the impact on the cost if the hash join is replaced by a merge join or a nested-loop join? Is the cost of the alternative query plan substantially higher compared to the QEP? In this context, a narrative that explains why the QEP is chosen by connecting its content with knowledge garnered from database textbooks will greatly benefit her learning.

Clearly, a learner-friendly tool that can facilitate exploration and visualization of the impact of various physical operators on a QEP can greatly enhance Doreen's comprehension of the query execution process. In this demonstration, we present a novel framework called MOCHA (iMpact of Operator CHoices visuAlizer) to aid user-friendly interaction and visualization of the impact of alternative physical operator choices on a selected QEP for a given SQL query. It is built on top of *PostgreSQL*. Given an SQL query and learner-specified *operator preferences* (*e.g.,* merge join, index scan), MOCHA automatically visualizes the impact of these choices on the selected QEP. This facilitates cost-based and structural comparison of the impact of different choices to aid learning. Furthermore, it generates a natural language-based explanation that goes beyond the conventional least-cost-based explanation to connect established knowledge related to usage scenarios of different physical operators that a learner has learnt from textbooks with the operators in a QEP. We demonstrate these features to showcase superior pedagogical support provided by MOCHA to database instructors and students.

2 SYSTEM OVERVIEW

Figure 2 shows the architecture of MOCHA and consists of the following components.

The Visual Interface (GUI) Module. The visual interface of MOCHA (Figure 3) enables a learner to view information related to the QEP of her input query and impact of alternative physical operators on it in a user-friendly manner. It consists of eight panels ($C1$-$C8$). The selection of an application-specific dataset (*e.g.,* TPC-H) and connection settings to the underlying RDBMS (*i.e., PostgreSQL*) are configured using the $C1$ panel. A learner inputs an SQL query in $C2$. The $C3$ and $C4$ panels allow her to specify various physical operators (*e.g.,* index scan, hash join) whose impact on the query she is interested to view. On clicking the *Query Plan* button ($C5$), MOCHA retrieves a set of AQPs based on these input operators and updates $C6$, $C7$, and $C8$ panels. Specifically, $C6$ provides a plausible natural language explanation of *why* the QEP is selected for the input query by connecting its content with existing knowledge. It also displays the cost of each AQP for the input query, enabling one to compare the cost of the QEP with these alternatives containing the learner-specified physical operator(s). The $C7$ and $C8$ panels allow one to visualize a query plan as a physical operator tree, which is an abstract representation of a query plan.

The Alternative Plan Choice Generator Module. To facilitate understanding of the query execution process, popular database textbooks typically discuss the impact of alternative choices of physical operators (*e.g.,* hash join vs merge join operators) on the estimated cost of a given QEP. This enables one to understand why certain physical operators are chosen for a QEP. For example, three operator trees involving three different join algorithms (*e.g.,* nested-loop join, hash join, and sort-merge join) may be compared to highlight the impact of the choice of a join algorithm on the query execution cost and subsequently on the selection of a QEP (*e.g.,* Examples 15.4-15.9 in Chapter 15 of [3]). In contrast to such knowledge garnered from a textbook, it is hard for a learner to *effortlessly* explore the impact of alternative operator choices on a QEP using an off-the-shelf RDBMS. Typically, these relational systems only expose the QEP of a query to its end users without giving them user-friendly access to information on the impact of alternative choices of physical operators.

We refer to a QEP whose one or more phyical operators are replaced by a set of alternative operators as an *alternative query plan (AQP)* or *alternative plan* for brevity. The *Alternative Plan Choice Generator* module is a core component of MOCHA and is responsible for retrieving the QEP as well as alternative plans involving alternative operator choices associated with a given SQL query in an RDBMS (*i.e., PostgreSQL*). Specifically, it exploits the *planner method configuration* (details are given in www.postgresql.org/docs/9.2/runtime-config-query.html#RUNTIME-CONFIG-QUERY-CONSTANTS) feature of *PostgreSQL* to generate AQPs based on a user input. The *configuration parameters* (*e.g., enable_hashjoin*, *enable_nestloop*) in this feature provide a way to enforce the query optimizer to choose a query plan with certain user-specified physical operators. Specifically, 11 parameters are exposed to a learner in $C3$. By default, all parameters are turned on during query processing. A query request is sent to *PostgreSQL* using the default settings to retrieve the QEP of a query.

In order to retrieve AQPs, a learner may check a subset of the configuration parameters based on the physical operators that she intend to view in these plans. When a check box is selected, the corresponding parameter is set to "true" (*e.g., SET enable_mergejoin = true*) in the query request. Otherwise, it is set to "false" (*e.g., SET enable_hashjoin = false*). MOCHA supports two modes (selected using $C4$) for generating alternative plans, namely, *single mode* and *multiple mode*. In *single mode*, MOCHA sends a query request to *PostgreSQL* in which the *unchecked* parameters on $C3$ are set to "false" to generate an AQP containing the operators corresponding to the checked parameters that are relevant to the processing of the query. In *multiple mode*, every checked parameter on $C3$ is either set to "true" or "false" to create all possible combinations of these parameters. MOCHA iterates through these combinations and sends corresponding query requests to *PostgreSQL*. Note that MOCHA only maintains all *distinct* alternative plans retrieved from these requests. Furthermore, it limits the number of parameters

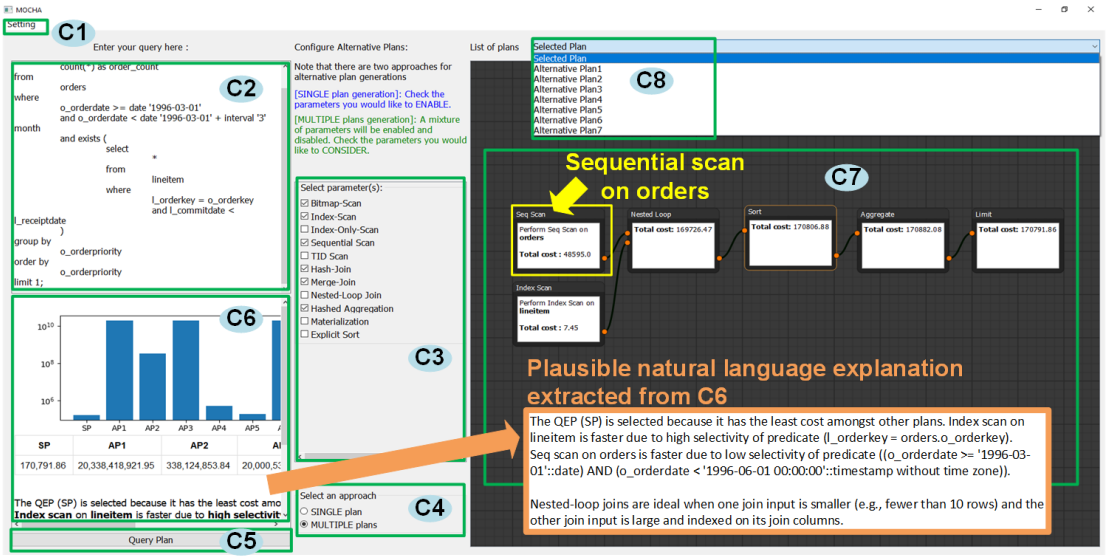

Figure 3: The visual interface of MOCHA.

that can be checked (set to 6 by default) to reduce the wait time for retrieving alternative plans. It is configurable, if required, through *C*1. For example, the checked configuration parameters in Figure 3 yield 64 combinations. These query requests, however, generate only 8 distinct query plans.

The cost of each retrieved plan is displayed at the top of *C*6 panel. A learner may use the slider to browse them and compare them with the cost of the QEP.

The Plan Diff Module. Given the physical operator trees of the QEP and an alternative plan, this module identifies the structural differences between them by extending the *DeepDiff* Python library (https://pypi.org/project/deepdiff/). In particular, nodes in an alternative query plan that are different from the QEP are highlighted by the *Plan Visualizer module* (see below). This allows a learner to quickly view the nodes in an alternative plan that are different from those in the QEP to aid learning. In particular, this module takes as input query plans (in JSON format) and cleans them to remove unnecessary details (*e.g.,* cost) so that the difference computation is based on the physical operators alone. Furthermore, only relevant operators are compared to this end (*e.g.,* a join operator is not compared with a scan operator).

The Plan Visualizer Module. This module is responsible for visualizing the retrieved plans as operator trees in *C*7. It takes a selected plan (through *C*8) as input and traverses it to retrieve information associated with each node (*e.g.,* node type, estimated cost, relation name, index condition, filter) and creates the corresponding visual node object. The visual nodes also contain information on intermediate tables where applicable. This module also invokes the *Plan Diff* module to identify the node differences w.r.t the QEP and display them in a color-coded format (red rectangles). Note that *C*7 displays one plan at a time.

The QEP Explainer Module. Learners often would like to understand why a QEP is selected for a given query that goes beyond the conventional least-cost reasoning. In particular, they would like to connect established knowledge related to usage scenarios of different physical operators that they have learnt in a database systems course with the selected operators in a QEP. The goal of this module is to construct such plausible explanations. Specifically, it aims to intuitively explain the reason for the choice of a join operator (*i.e.,* merge join, hash join, nested-loop join) and scans (*i.e.,* index scan, sequential scan) in a QEP. For instance, (a) index scan is the optimal access path for low selectivity whereas sequential scan performs better in high selectivity [2]; (b) merge join is preferred if the join inputs are large and are sorted on their join attributes [1]; (c) nested-loop join is ideal when one join input is small (*e.g.,* fewer than 10 rows) and the other join input is large and indexed on its join attributes [1]; (d) hash join is efficient for processing large, unsorted and non-indexed inputs compared to other join types [1]. To this end, we manually extract usage scenarios of different physical operators from the relevant literature. This is feasible since there is a small number of physical operators in *PostgreSQL*. Then a set of documents containing these usage scenarios is indexed using an inverted index where each document is associated with a single physical operator.

The *QEP Explainer* scans the retrieved QEP to identify these operators and retrieves associated predicates and join conditions, if any. The text explanation is then generated for an operator by utilizing a rule-based template, the inverted index to retrieve corresponding usage scenario, and database statistics information (*e.g.,* selectivity). The generated explanation is displayed in *C*6. For example, the selected QEP in Figure 3 uses index scan on the lineitem table as it is faster due to the high selectivity of the predicate (*i.e.,* l_orderkey = orders.o_orderkey).

3 RELATED SYSTEMS

Several interesting features of query optimizers have been demonstrated in major conference venues [4, 8, 9]. However, to the best of our knowledge, there has been no prior demonstration on the visualization and exploration of the impact of alternative choices of different physical operators on a QEP to support database education.

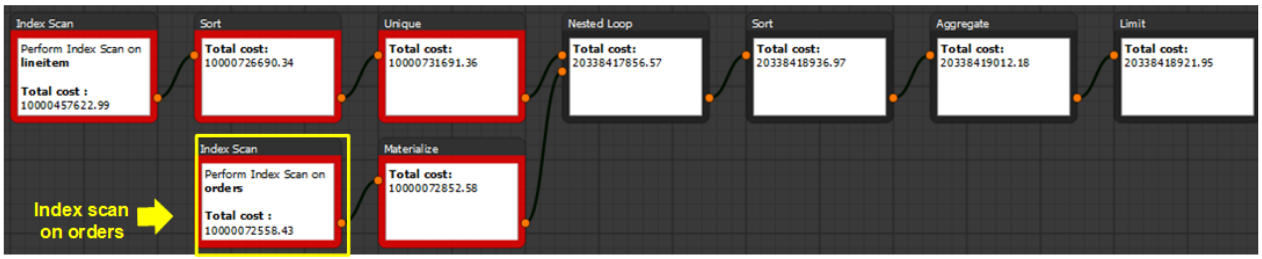

Figure 4: An alternative query plan (AQP).

Natural language interfaces to relational databases have been studied for several decades. Given a logically complex English language sentence as query input, the goal of majority of these work is to translate it to SQL [6]. The *QEP Explainer* module of MOCHA complements these efforts by providing a plausible natural language explanation of a QEP. Most germane to this module are recent efforts in [7, 10, 11] to generate natural language *descriptions* of QEPs to support database education. In particular, the goal is to describe the execution steps of a QEP in a natural language. [7] also supports a *natural language question answering* system that allows a user to seek answers to a variety of concepts and features associated with a QEP. In contrast, MOCHA's *QEP Explainer* module does not focus on describing the steps in a QEP. Instead, it intuitively explains why the key operators in a QEP are chosen by connecting them to established knowledge in the literature.

4 DEMONSTRATION OVERVIEW

MOCHA is implemented using Python 3 and *PostgreSQL*. Our demonstration will make use of the TPC-H decision support benchmark and IMDB data set (https://relational.fit.cvut.cz/dataset/IMDb). Users can pose their own ad-hoc queries on these datasets using MOCHA.

The goal of our demonstration is to allow the audience to experience the following interactive features of MOCHA. A **short video** to illustrate the main features of MOCHA using example use cases is available at https://youtu.be/jJJ25LH6DLA. MOCHA is available at https://howardlee.cn/mocha/.

Single alternative plan visualization. Through the MOCHA GUI, an audience can input an SQL query in *C*2 (*e.g., Query 4* in TPC-H), select the configuration parameters of interest in *C*3, choose the *single plan* in *C*4, and click the *Query Plan* button. This will trigger MOCHA to send two query requests to *PostgreSQL* where the first request is a query with default parameter settings and the second request is one with unchecked parameters set to "false" (*e.g., SET enable_indexscan = false; SET enable_hashjoin = false; SET enable_mergejoin = false;*). The former request retrieves the QEP (*SP*) whereas the latter request yields an alternative plan (*AP*1). One can visualize these plans in *C*7. In particular, structural differences between *SP* and *AP*1 are color-coded for easy reference. Lastly, the total costs of *SP* (170791.86) and *AP*1 (10000537174.53) are displayed in *C*6 for ease of comparison.

Multiple alternative plans visualization. An audience can select the *multiple mode* in *C*4 to generate a list of combinations of the selected parameters to retrieve alternative plans. Upon clicking of the *Query Plan*, the query with default parameter settings is first submitted to *PostgreSQL* to retrieve the QEP. Then, the query with each parameter combination is submitted in turn to retrieve AQPs. Recall that only distinct plans are maintained in MOCHA. *C*6 is updated accordingly. In the example in Figure 3, the costliest alternative plan (Figure 4) is *AP*1 with the total cost of 20338418921.95. The main difference between *AP*1 and *SP* (*i.e.,* QEP) is that *SP* performs a sequential scan on orders whereas in *AP*1, index scan is performed on orders, followed by a sort (can be visualized by invoking the *Plan Diff* module). The low selectivity of the predicate on o_orderdate makes the sequential scan a preferred choice.

Explanation of QEP. Lastly, an audience can view the generated explanation in *C*6 of the QEP that connects established knowledge in the literature to the selected operators in the QEP. An example is depicted in Figure 3.

ACKNOWLEDGMENTS

We thank our collaborators Baochao Xu and Hui Li for hosting the MOCHA system in Xidian University, China.

REFERENCES

[1] Advanced query tuning concepts. https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms191426(v=sql.105)?redirectedfrom=MSDN, last accessed on 15th June, 2022.
[2] R. Borovica-Gajic, S. Idreos, A. Ailamaki, M. Zukowski, C. Fraser. Smooth scan: robust access path selection without cardinality estimation. *The VLDB Journal*, 27(4):521-545, 2018.
[3] H. Garcia-Molina, J. D. Ullman, J. Widom. Database Systems: The Complete Book. *Prentice-Hall*, 2002.
[4] J. R. Haritsa. The Picasso Database Query Optimizer Visualizer. *Proceedings of the VLDB Endowment*, 3(2), 1517-1520, 2010.
[5] Z. Ives, J. Gehrke, J. Giceva, A. Kumar, R. Pottinger. VLDB Panel Summary: "The Future of Data(base) Education: Is the Cow Book Dead?". *SIGMOD Record*, 50(3), 23-26, 2021.
[6] H. Kim, B.-H. So, W.-Shin Han, Hongrae Lee. Natural Language to SQL: Where Are We Today? *Proceedings of the VLDB Endowment*, 13(10), 1737-1750, 2020.
[7] S. Liu, S. S. Bhowmick, W. Zhang, S. Wang, W. Huang, S. R. Joty. NEURON: Query Optimization Meets Natural Language Processing For Augmenting Database Education. *In Proceedings of the 2019 ACM SIGMOD International Conference on Management of Data*, 1953-1956, ACM, 2019.
[8] H. Pirk, O. R. Moll, S. Madden. What Makes a Good Physical plan?: Experiencing Hardware-Conscious Query Optimization with Candomble. *In Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data*, 2149-2152, ACM, 2016.
[9] D. Scheibli, C. Dinse, A. Boehm. QE3D: Interactive Visualization and Exploration of Complex, Distributed Query Plans. *In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 877-881, ACM, 2015.
[10] W. Wang, S. S. Bhowmick, H. Li, S. Joty, S. Liu, P. Chen. Towards Enhancing Database Education: Natural Language Generation Meets Query Execution Plans. *In Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data*, 1933-1945, ACM, 2021.
[11] P. Chen, H. Li, S. S. Bhowmick, S. R. Joty, W. Wang. LANTERN: Boredom-conscious Natural Language Description Generation of Query Execution Plans for Database Education. *In Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data*, 2413-2416, ACM, 2022.