Using MDE for Teaching Database Query Optimizer

Abdelkader Ouared[a] and Abdelhafid Chadli[b]
Ibn Khaldoun University (UIK), Tiaret, Algeria

Keywords: Database Query Optimizer Learning, Using MDE in Teaching, Metamodeling, Domain-specific Language.

Abstract: Query optimization is considered as an important part of Data Base Management Systems (DBMS) and plays a major role in database research community (e.g. SQL, Spark, Map-reduce). Generally, this optimization is done using Cost Base Optimization (CBO), which is difficult to understand due to the complexity of platform, database, query and DBMS. Moreover, database query optimizer is usually a very complex process, with difficult concepts depending on the behaviour of the query engine of DBMS. Therefore, users (e.g. novice user, developer, DB administrator) have difficulties to understand and build a mental image of database query optimizer in order to produce more efficient queries. In this paper, we propose a Framework based on the model-driven engineering (MDE) paradigm to facilitate understanding and improving query performance. Indeed, MDE approach has been proven useful for developing new software applications, and its adoption for a teaching perspective presents a major challenge. We illustrate our proposal with use case and proof-of-concept prototype for the whole provided process.

1 INTRODUCTION

Generally speaking, the query optimizer is viewed as the bottleneck of database application performance. Recently several calls and demands have been raised to make this design much easier, by teaching DBMS architecture and query optimization process (Scherzinger, 2019), (Pavlo et al., 2017). Query Optimizer process is difficult to understand since it includes several parameters belonging to databases, platforms, DBMS, queries, etc. Furthermore, in all existing query optimizers, users are systematically out of the process loop. To remedy this situation, a variety of query optimizer (SQL, Map-reduce, etc.) might be used or taught for different purposes such as including users to interact with the optimizing process.

1.1 Problem Statement

As stated above, query optimizers are considered as an important part of DBMS and plays a major role in database research community (e.g. (Siddiqui and Other, 2020; Gallinucci and Golfarelli, 2019)). This component is one of the most important quality indicators that companies are looking for to choose their appropriate query process in DBMS, since they estimate metrics associated to non-functional requirements (e.g., query response time, Quality of service, energy consumption). The DBMS must then design an execution strategy or query plan to retrieve the query results from the database files. A query has many possible execution strategies, and the process of choosing a suitable one to handle an SQL query is known as Cost Based Optimization (CBO) (Jarke and Koch, 1984). Generally, this optimization is done using CBO, which is difficult to understand due to the complexity of platform, database, query and DBMS. Moreover, Query Optimizers are usually seen as black box systems, and depend on cardinality estimates, plan properties, and specialized cost formulas for each relational algebra operator in an execution plan. Consequently, database users have difficulties to build a mental image on query optimization process and how to improve query performance. For example, how to use a special hint (e.g. HASH) in a way to dictate the optimizer a different retrieval path rather than that one calculated by the optimizer (e.g. rewriting the query differently so as to influence the default query execution). Indeed, one of the most challenging aspects of teaching advanced database in undergraduate computer programs is the query optimizer (Liu et al., 2019). To address the lack of query performance undergraduate course teaching, we present in this paper

[a] https://orcid.org/0000-0003-4257-0522
[b] https://orcid.org/0000-0003-0559-1439

Ouared, A. and Chadli, A.
Using MDE for Teaching Database Query Optimizer.
DOI: 10.5220/0010535105290536
In *Proceedings of the 16th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2021)*, pages 529-536
ISBN: 978-989-758-508-1

an MDE-based framework to facilitate understanding and improving query performance. We aim to answer the following research question.

RQ: *is it possible to build an MDE-based conceptual framework to help database users understand the SQL Cost Based Optimizer process*?

1.2 Paper Contributions

There has been a lot of work in the query optimization field. However, previous works have not focused on facilitating the learning experience to be rewarding for students and professionals so that they better understand how the system works. In this paper, we focus on solving these difficulties by proposing a framework for assisting users to analyze and better cope with CBO difficulties on Relational DBMS. We propose a framework called CF-CBO (Conceptual Framework for teaching CBO). Our framework explicit Query Optimizer at a high level of abstraction and provides students with CBO-processing mental image to facilitate understanding and improving query performance. We first propose a domain specific language (DSL), called OptDSL dedicated to describe Database Query Optimizer. Second, we describe the methodology associated with our framework to explain how the system works. Finally, we illustrate our Tool Support.

1.3 Paper Outline

The paper is structured as follows. Section 2 presents the background and discusses the related work. In Section 3, we present our proposed approach. Section 4 highlights the implementation of our proposition. Finally, Section 5 concludes this paper.

2 BACKGROUND

In order to make this research self-contained and straightforward, this section introduces the notions of Query Optimization for Database Systems.

2.1 Query Optimization Domain Description

Let us define a query plan as a tree $P = (O, E)$ (see Figure 1 (b)), where the set of nodes O of the tree P represents the operations (i.e. selection in relational algebra), and the set of edges E denotes the precedence between two operations. More precisely, a plan P_j is a set of relational algebra operators (e.g., restriction, projection, join, scan, sort), $P_j = \{O_{1j}, \ldots, O_{nj}\}$. Each $O_i \in P_j$ is annotated with a quadruplet $O_i = (Imp_j, T_j, C_j, ProP_i)$ in which Imp_j is the implementation algorithm for each physical query operator (e.g. *Nested-loops-join algorithm* to implement a *Join* Operation) , T_j is the set of associated input database objects such as tables, indexes, materialized views etc., C_j is the set of options used to execute the operation, like adding the buffer option, and $ProP_i$ is the Programming Paradigm (*Sequential*, *MapReduce*, *etc*.). Each directed edge $e \in E$ is an ordered pair of vertices defined as $e = (i, j)$ and connects the operation i to the operation j when the execution of i directly precedes that of j. A global plan P_j may possibly aggregate several subplans $SP_{j_1}...SP_{j_n}$ to be executed on the database. Figure 1 shows the query plan for TPC-H [1] query 9 depicted as a tree structure based upon the logical operators in the query. Each node is responsible for executing one SQL operator in the query, like sort and aggregation.

The Query Optimizer converts (or *rewrites*) logical requests, expressed in SQL language, into physical plans in order to find a good execution plan in terms of execution costs. Hence, the Query Optimizer uses two strategies (Rule-Based Optimization, Cost-Based Optimization) to associate several candidate execution plans with a given SQL query and chooses the best one. The query optimizer (Jarke and Koch, 1984), uses the CBO approach by means of cost models (Manegold et al., 2002) to process the cost of each operation in the query. For example, CBO strategy can be used to estimate I/O and CPU costs of each execution plan in order to choose the best cost by making a calculation of the database statistics. The Query Optimizer as three main components, namely, *the plan generator*, *the performance cost calculator* and *the plan evaluator*. Figure 2 shows the workflow between these three parts. First, the plan generator produces the candidate plans of a given query, then performance cost calculator estimates the cost of each operation in the query plan, and finally the plan evaluator returns the optimal query plan.

2.2 Motivating Example

Let us consider a real scenario, where a database user wants to optimize an SQL query. A typical query optimizer relies on host hardware information and database statistics (*called metadata*) to find the cheapest execution plan for an SQL statement. In Figure 1,we depict an example taken from the TPC-H benchmark schema to optimize the total costs. First, the user can visualize the query plan

[1] http://www.tpc.org/tpch/

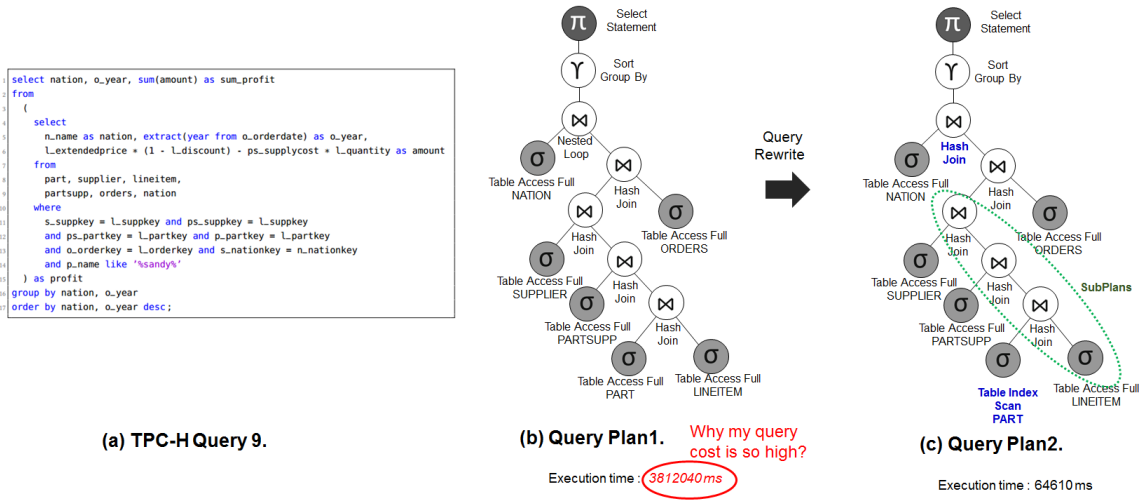


Figure 1: Two query plans of TPC-H query 9.

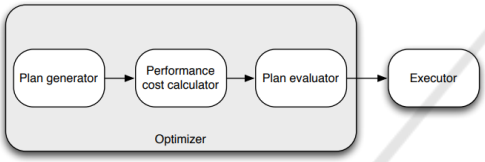


Figure 2: The query optimizer components workflow.

by using for instance the `EXPLAIN ANALYZE` instruction, then he can rewrite the query by forcing the optimizer through query hints (`e.g. SET enable hashjoin=off`) to determine which physical relational operator should be used, he can also choose the physical structure, like index, in order to improve the cost. For example, we can see in Figure 1 (a,b) that when nested loop join has been replaced with the hash join and the full scan been replaced with the index scan, as consequence, the query performance related to the CPU overhead and disk I/O was improved (Execution Time decrease from: 3812,040 seconds ⇒ 64,610 seconds) (**is 60x faster**). From these results, it is concluded that choosing of hints, like join operation, and physical structure, like index, has a significant impact on the cost of queries but not explained or justified.

2.3 Existing CBO Tools in Academic and Industrial Areas

There is a long line of studies on cost based optimizers for database users (e.g. SQL/SparkSQL,Spark, SQL-on-Hadoop engine, Map-reduce, Spark, Flink, SQL) because it is considered as an important part of DBMS and plays a major role in database research community. From the industry and academia, there has been much research on CBO problems, such as the automatic selection of materialized views and indexes selection (Agarawal and Others, 2000). In addition, most modern databases come with designer tools, e.g., Tuning Wizard in Microsoft SQL Server (Chaudhuri and Narasayya, 1998), Teradata's Index Wizard (Brown et al., 2009), and Oracle's SQL Tuning Adviser (Bhagat and Shukla, 2014). However, these tools optimize query without proposing a mental image of this process. By examining the database design life cycle, we found that computer science education follows the traditional 3-phase design: conceptual, logical and physical. However, a revue of educational solutions for database design showed that topics in educational domain include only database design (e.g. (Keberle and Utkin, 2012; Al-Dmour, 2010)), relational algebra and SQL (e.g. by interacting tools like QueryVis (Leventidis and Other, 2020), concurrency control in DBMS (e.g. (Allenstein and Others, 2008; Scherzinger, 2019)), Database Tuning (Almeida and Others, 2019) and teaching DBMS architecture (Sciore, 2007; Galaktionov and Chernishev, 2019). Furthermore, we found in the literature a lot of work that is interested in CBO related to SQL queries. For example, academic and industrial database admin tools (Parinda, IndexStore, etc.) use CBO by iterative invocations of cost evaluation module, but users are keep out the loop of the optimizing process. These tools provide optimal performance cost with less knowledge about how to optimize query plans and no effort is made to teach CBO basic knowledge to students, doctoral students and professionals.

3 OUR FRAMEWORK

This section is devoted to present the CF-CBO *Conceptual Framework for Cost Based Optimization*. CF-CBO is intended to move the Database Query Optimizer process from depth complex details to a high level of abstraction so as to facilitate the learning process. We first formalize relevant foundations of the CF-CBO Framework using a DSL, called OptDSL, dedicated to Query Optimizer domain. Then, we give an overview of the framework workflow that explains the whole provided process.

3.1 CF-CBO Overview

Since the paper contributions have led to the development of the CF-CBO framework, we first give an overview of the capabilities that it provides. Our Framework is based on MDE paradigm that focuses on the use of models and model transformations to raise the level of abstraction and automation in query optimizer. CF-CBO offers the following services: **(i) Service 1:** Visual plan construction of a given query that is expressed in OptDSL. The goal of optimization is to minimize the cost of a given SQL Query by additional physical information for each relational algebra operation. **(ii) Service 2:** Query Rewriting from any query plan expressed by Service 1. The generated SQL code have to be compliant with DBMS. **(iii) Service 3:** This service helps users to combine their modifications to be included in a global query plan. User interactions with OptDSL metamodel instance (Add, Remove, etc.) and feedback are saved in event logs and used later to provide users with hint suggestions to help choosing different relational algebra operations (e.g. Join Operator type) according to algorithms implementation (e.g. NestedLoop-Join, Hash-Join, etc.)

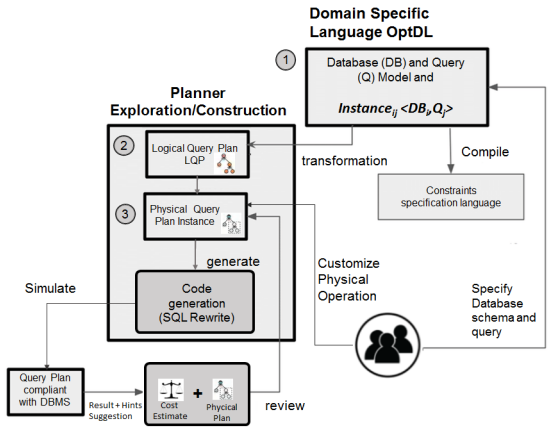


Figure 3: Overview of the CF-CBO Framework.

Figure 3 shows all these services and how they communicate with each other. Indeed, Service 1 is based on the OptDSL language (presented in Section 3.2)), it allows expressing the query plan as an instance of the OptDSL metamodel. That will help to edit, display, insert, and remove query operators under the same formalism. In this case, Service 3 is required. That is, every user can use Service 3 not only for visual query exploration, but also to obtain a complete user documentation as metamodeling hierarchy that defines the query optimizer components and properties thanks to the syntax and semantics of our OptDSL. The benefit is that during the optimizing process, we assume that user understands OptDSL language since it contains a set of concepts, which are common usually in CBO. Another important point is that user can generate a transformation script in SQL once a draft version of a query plan is obtained, rather than rewriting the query manually, to avoid statement errors. Service 2 provides this transformation automatically based on our OptDSL metamodel. The CF-CBO framework is in charge of generating the source code of the targeted DBMS (e.g. PostgreSQL). In the following, we will focus on the cornerstone elements of Service 1, Service 2 and Service 3.

3.2 Core Elements of CF-CBO

The OptDSL language is the core element of CF-CBO, dedicated to CBO. An instance of the OptDSL language can represent a user need, which consists in understanding/improving a query performance. Users can interact with an instance of the OptDSL language to understand/improve a query performance. In this case, the user has to specify the context of his system (i.e. database instance, query and platform) on which he needs help. By instantiating OptDSL language, we can also get a detailed relational database query plan. Hereafter, we focus on core elements of OptDSL metamodel and their semantics. Figure 4 depicts elements that are dedicated to express the optimizer and its characteristics at a high-level of abstraction. The root element of the metamodel is the QueryPlan class, each QueryPlan instance consists of a Database class instance, a Query class instance, a Platform class instance, an Optimization-Structure class instance, a Hints class instance, an Operation class instance and a CostModel class instance. An instance of the *Database* class is composed of conceptual entities and their attributes. In addition, links between entities are also represented via associations. We also represent several semantic restrictions such as primary and foreign keys. An instance of the *Query* class takes as input a set of con-

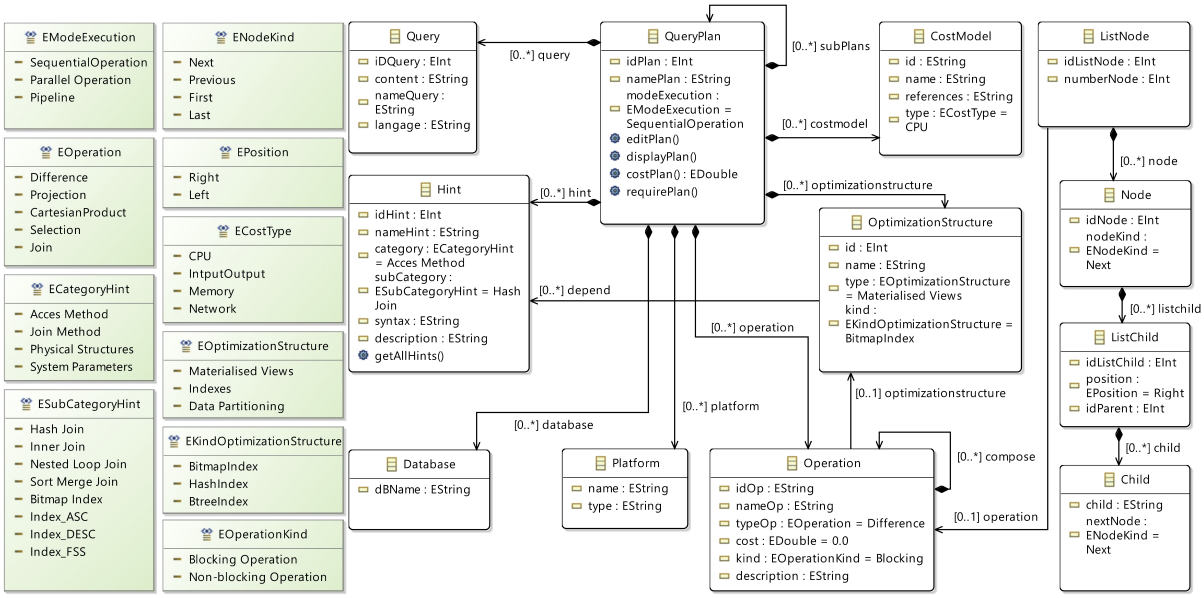


Figure 4: Excerpt of OptDSL Meta-Model: Core Entities.

cepts, which are used to perform a set of algebra operations (join, union, etc.). Furthermore, query optimizer can use access method such as *B-tree index* or *in-memory access method*. An instance of the Platform class includes the deployment architecture of the database such as distributed or parallel database systems, database clusters, or cloud environments. An instance of *Cost Model* class is defined as a function and corresponds to a given metric of an elementary query operation, it is composed of a metric (an instance of *Metric* class), a context (an instance of *Context* class) and a cost function (an instance of *CostFunction* class). Also, every instance of the *CostModel* class has at least one cost type (for more details, refer to our previous work (Ouared et al., 2016)).

3.3 The CF-CBO Process

The proposed approach is a multi-step process that primarily consists of 3 phases as depicted in Figure 3: (1) Plan Exploration (2) Visual Query Construction, and (3) Query Plan Rewrite. In the following, we detail this process systematically.

3.3.1 Plan Exploration

Plan Exploration helps users to understand and explore a query plan by showing the query plan/subplan, the order of operations, and used hints. This assistance allows users browsing the knowledge of CBO concepts (e.g. physical operations like *scan*, *sort*, *join*...), identifying correctly the order of operations, and accessing SubQuery and Query Plan Methods. Furthermore, *Plan exploration* provides a complete user documentation since the metamodeling hierarchy defines all components of the query optimizer and the properties of these components. Thanks to the syntax and semantics of our Query Optimizer DSL, our metamodel relies on concepts categorization that will play a key role in the query optimizer's process learning (see Section 3.2)). One should use the interface in order to express the query and visualize the plan to be analyzed. One should use the interface in order to express the query and visualize the plan to be analyzed. Figure 5 shows an example of the Q9 query plan (c.f. Section 2 see Figure 1)) with seven subplans using DSL parser as shown in Figure 5.

3.3.2 Visual Query Construction

The presence of a design language that allows browsing the query plans, enriching, editing, validating, transforming them, etc. represents a valuable asset for database users. Users can use our design tool to select the required hints and to identify the required physical configuration parameters that reduce the query cost. In addition, users are required to decide about choice of different relational algebra operations depending on their implementation algorithms. For example, use of special hint (e.g. HASH JOIN), rewrite queries (e.g. remove GROUP BY clause), add/remove physical structures (such as materialized views, indexes, or a combination of these). In addition, users can display and customize various internal performance parame-

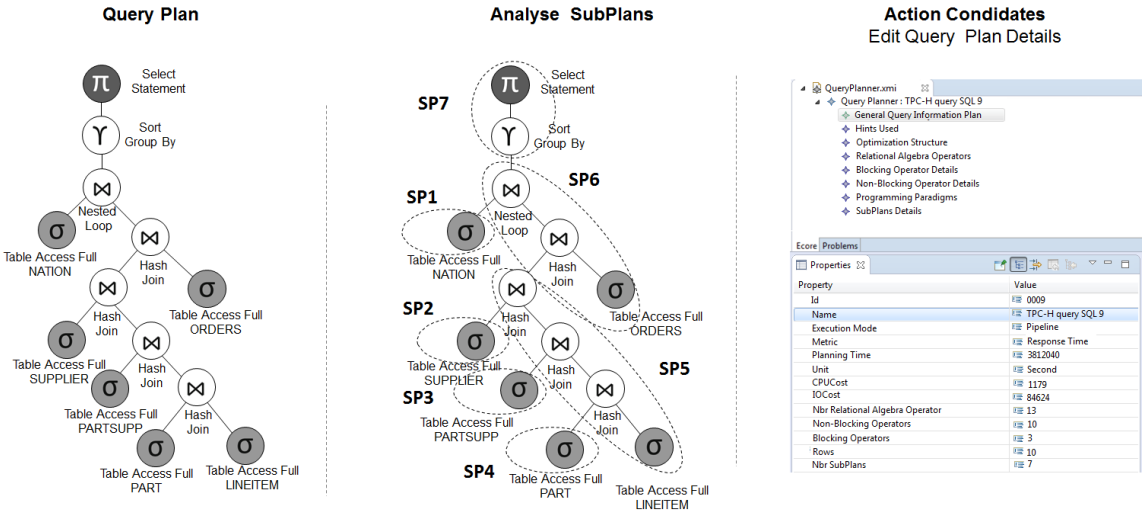


Figure 5: Segmentation of query *Q9* into Subplans.

ters like *Shared SQL Pool*, *Redo Log Buffer* and other information needed in the query optimization.

3.3.3 Query Plan Rewrite

Every query plan instance is generated according to OptDSL. For that, a set of structural rules have been injected in the OptDSL. These rules are expressed as OCL (Object Constraint Language) invariants. Listing 1 is an example of a structural rule. This example means that all implementation type operations and their programming paradigm, that are query plan inputs, have to be referenced as `OperationType` instances in the `ProgrammingModelType` instance of the operation class.

```
Class Operation
self.Implementation.type->includesAll(self.
    EImplementationType)
and self.ProgrammingModel.type->includesAll(self
    .EProgrammingModel)
```

Listing 1: An OCL structural rule.

In order to transform any query plan that is expressed in OptDSL language into PostgreSQL, we have developed a code generator based on MDE settings using the model-to-text transformation, since our objective is to obtain the query physical plan. The implementation relies on the utilization of *Acceleo*. Listing 2 shows the generated query plan of the example presented in Figure 1.

```
Sort  (cost=410317.35..410317.37 rows=6 width
    =25) (actual time=10424.815..10424.816 rows
    =4 loops=1)
  Sort Key: l_returnflag, l_linestatus
  Sort Method: quicksort  Memory: 25kB
  ->  HashAggregate  (cost=410317.11..410317.27
      rows=6 width=25) (actual time
      =10424.769..10424.775 rows=4 loops=1)
       Group Key: l_returnflag, l_linestatus
       ....
Planning time: 0.380 ms
Execution time: 10424.894 ms
(10 rows)
```

Listing 2: Example of Displaying Execution Plans.

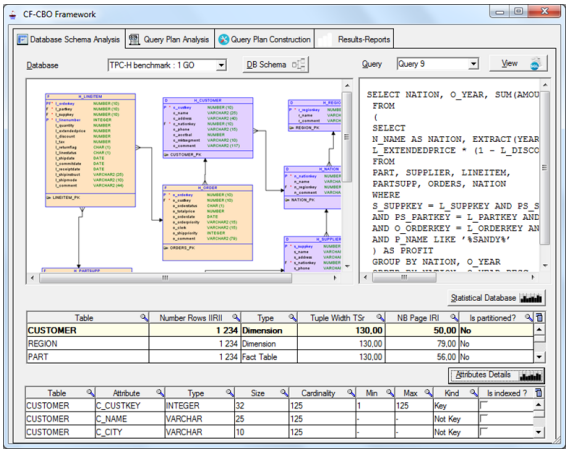
4 PROOF OF CONCEPT

To stress our approach and to proof how it is useful and helpful, this section is devoted to present a global usage scenario of the CF-CBO framework. CF-CBO has been implemented for academic requirements based on PostgreSQL[2], mostly because it is open source. To ease the understanding, we provide the URL of a demonstration video of the CF-CBO framework[3]. In this section, we describe the functionalities of CF-CBO which is composed of two parts, the backend which contains the DBMS and the frontend part that is represented by the graphical user interface. The component modules are listed below.

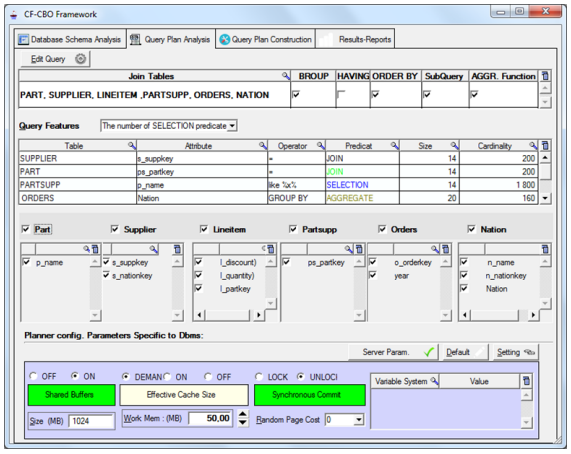
The CBO Learning support is divided into four parts: (1) Database Schema Analysis, (2) Query

[2]http://www.postgresql.org/

[3]The demonstration video of CF-CBO is available at: https://www.youtube.com/watch?v=OGn3mf4s2aQ

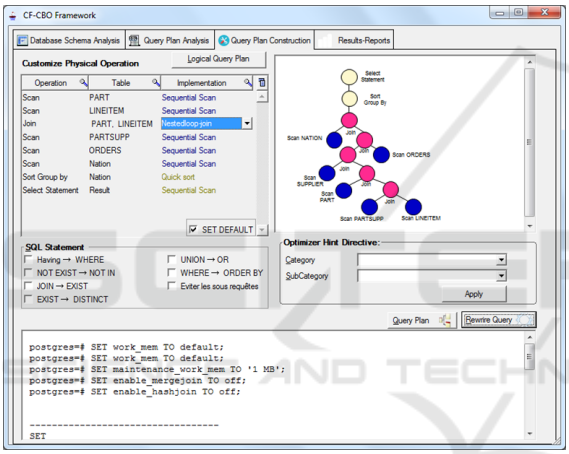


(a) Database Schema Analysis

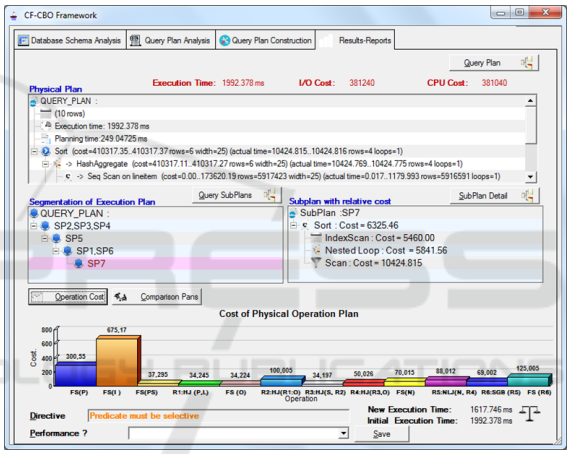


(b) Query Plan Analysis

Figure 6: CF-CBO main GUI and its component module panels (A).



(a) Query Plan Construction



(b) Results-Reports

Figure 7: CF-CBO main GUI and its component module panels (B).

Analysis, (3) Query Plan Construction and (4) Results-Reports and Plan Exploration. In the beginning, our tool displays a configuration form that includes database connection information. When the user is connected, he has first to select a database schema (e.g. TPC-H schema database) among many ones and then selects a query of TPC-H benchmark [4] to optimize (cf. Figure 6 ⓐ). The Database Schema Analysis panel assists users in analyzing information about database statistics. Figures 6 and 7 show the CF-CBO main GUI and its component module panels. The Query Plan Analysis panel helps users to get information about the query structure (Selection/Join predicate, aggregation functions, etc. These information are required to improve query performance (cf. Figure 6 ⓑ). In Query Plan Construction (cf. Figure 7 ⓐ), the system displays three informational panels. On the top right, our tool displays the logical query plan indicating the algebra operators' execution order to enable the user making a global understanding on query result. On the top left, the user can customize algorithm implementation for each relational algebra operator in the query plan. In addition, user can get information about database server parameters and customize each value appropriately such as *the Shared SQL Pool* and *system page size*. Finally, the user can select the hints that are organized into categories (e.g. Access Paths) and subcategories (e.g. FULL Scan). The output of this stage is a rewritten SQL query with automatic code generation from high-level abstraction description language. In panel Results-Reports (cf. Figure 7 ⓑ), the user can verify if the execution time of the new query plan is higher cost than the initial cost and tries to refine the query plan by return-

[4] http://www.tpc.org/tpch/

ing to panel three. In order to assess the user's query performance skill, we employed a checklist as educational strategy to get the participants' feedback by selecting the correct or best response from a provided list (as shown in Figure 7 ⓑ). The system prompts the user to answer some questions, like *why query has a lower/upper cost*? with several answers such as "Is the data for this query benefit from cached data" etc. The interactions and feedback of users are saved in the *the event logs*. This latter indicates users' performance based on collected data from users' actions during interaction with CF-CBO.

5 CONCLUSION

This paper presents a Conceptual and Methodological Framework to explain Database Query Optimizer to users dealing with query performance and to help transferring the domain knowledge effectively to that users. The work is motivated by the complexity of the CBO entity compared to its cardinality estimates, plan properties, database operations and its algorithms. Our approach supports that MDE can be applied to a real problem in advanced courses of database, in particular, the cost based optimization. The main contributions of this work are as follows. First, (i) we built a metamodel to explicit Database Query Optimizer knowledge and to provide a domain vocabulary that helps learning query plan, understanding and combining hints, and customizing query plan. Second, (ii) we developed an MDE based process that offers different bricks (operations, hints) that are required in CBO to facilitate query plan optimization. Finally, (iii) we developed the tool support of the whole approach. Currently, we are evaluating how the CF-CBO can significantly increase efficiency and effectiveness of the students understanding in the query optimization domain. Nevertheless, this work opens several directions of further research. First, we are studying to explore more intensively DBMS inside in order to deliver a more informative instruction on database query optimization at a high level of abstraction. Second, we project to develop a rule-based errors diagnosis related to users selection of required hints to enhance CBO learnability.

REFERENCES

Agarawal, S. and Others (2000). Automated selection of materialized views and indexes for Sql databses. In *Proceedings of 26th International Conference on VLDB, Cairo, Egypt*, pages 191–207.

Al-Dmour, A. (2010). A cognitive apprenticeship based approach to teaching relational database analysis and design. *Journal of Information & Computational Science*, 7(12):2495–2502.

Allenstein, B. and Others (2008). A query simulation system to illustrate database query execution. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pages 493–497.

Almeida, A. C. and Others (2019). An ontological perspective for database tuning heuristics. In *International Conference on ER*, pages 240–254. Springer.

Bhagat, M. and Shukla, S. (2014). Tuning the tpc-c benchmark using oracle 10 g. *IJEMR*, 4(5):179–182.

Brown, D. P., Chaware, J., and Koppuravuri, M. (2009). Index selection in a database system. US Patent 7,499,907.

Chaudhuri, S. and Narasayya, V. (1998). Autoadmin "what-if" index analysis utility. *ACM SIGMOD Record*, 27(2):367–378.

Galaktionov, V. and Chernishev, G. (2019). Designing a DBMS development course with automatic assignment evaluation. In *Fourth Conference on SEIM-2019 (full papers)*, volume 8, page 48.

Gallinucci, E. and Golfarelli, M. (2019). Sparktune: tuning spark Sql through query cost modeling. In *EDBT*, pages 546–549.

Jarke, M. and Koch, J. (1984). Query optimization in database systems. *ACM Computing surveys (CsUR)*, 16(2):111–152.

Keberle, N. and Utkin, I. V. (2012). Teaching conceptual modeling in er: Chen worlds. In *ICTERI*, pages 222–227.

Leventidis, A. and Other (2020). Queryvis: Logic-based diagrams help users understand complicated Sql queries faster. In *Proceedings of the 2020 ACM SIGMOD*, pages 2303–2318.

Liu, S., Bhowmick, S. S., Zhang, W., Wang, S., Huang, W., and Joty, S. (2019). Neuron: Query execution plan meets natural language processing for augmenting DB education. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1953–1956.

Manegold, S., Boncz, P., and Kersten, M. L. (2002). Generic database cost models for hierarchical memory systems. In *VLDB*, pages 191–202.

Ouared, A., Ouhammou, Y., and Bellatreche, L. (2016). Costdl: a cost models description language for performance metrics in database. In *2016 21st International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 187–190. IEEE.

Pavlo, A. et al. (2017). Self-driving database management systems. In *CIDR*, volume 4, page 1.

Scherzinger, S. (2019). Have your Students build their own Mini Hive in just Eight Weeks. In *LWDA*, pages 38–41.

Sciore, E. (2007). Simpledb: a simple java-based multiuser syst for teaching database internals. *ACM SIGCSE Bulletin*, 39(1):561–565.

Siddiqui, T. and Other (2020). Cost models for big data query processing: Learning, retrofitting, and our findings. In *Proceedings of the 2020 ACM SIGMOD*, pages 99–113.