An Ontological Perspective for Database Tuning Heuristics

Ana Carolina Almeida[1](✉), Maria Luiza M. Campos[2], Fernanda Baião[3], Sergio Lifschitz[3], Rafael P. de Oliveira[3], and Daniel Schwabe[3]

[1] State University of Rio de Janeiro (UERJ), Rio de Janeiro, RJ, Brazil
ana.almeida@ime.uerj.br

[2] Federal University of Rio de Janeiro (UFRJ), Rio de Janeiro, RJ, Brazil
mluiza@ppgi.ufrj.br

[3] Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Rio de Janeiro, RJ, Brazil
fbaiao@puc-rio.br,
{sergio,rpoliveira,dschwabe}@inf.puc-rio.br

Abstract. Database tuning is a complex task, involving technology-specific concepts. Although they seem to share a common meaning, there are very specific implementations across different DBMSs vendors and particular releases. Database tuning also involves parameters that are often adjusted empirically based on rules of thumb. Moreover, the intricate relationships among these parameters often pose a contradictory impact on the overall performance improvement goal. Nevertheless, the literature – and practice – on this topic defines a set of heuristics followed by DBAs, which are implemented by the available tuning tools in different ways for specific DBMSs. In this paper, we argue that a semantic support for the implementation of tuning heuristics is crucial for providing DBAs with a higher-level conceptualization, unburdening them from worrying about internal implementations of data access structures in distinct platforms. Our proposal encompasses a set of formally-defined rules based on an ontology, enabling DBAs to define new configuration parameters and to assess the application of tuning heuristics at a conceptual level. We illustrate this proposal with two use case scenarios that show the advantages of this semantic support for the definition and execution of sophisticated DB tuning heuristics, involving hypothetical indexes and what-if situations for relational databases.

Keywords: Database tuning · Ontology · SWRL · Heuristics

1 Introduction

Computational systems are increasingly ubiquitous, producing and consuming large amounts of data. As a consequence, these systems pose a demand for higher performance, especially for lower response time and increased throughput, pushing Database Management Systems (DBMSs) to higher levels of functionalities and control. Database (DB) tuning strategies address this need by supporting the configuration of the physical design of the DB towards improving performance. DB tuning involves decisions upon the creation and maintenance of indexes, materialized views, data replication, partitioning, query rewriting, among others. These strategies constitute heuristics for improving the performance of the applications that access this DB [1]. As any other heuristics, DB tuning heuristics may be implemented and automated to assist their users (in this case, the Database Administrator, or DBA) in achieving a goal (i.e., improving performance), and obtaining successful results in most cases [1]. The tools that implement DB tuning heuristics can be semi-automatic (when the DBA makes the final decision on tuning the DB based on the suggestions of the heuristics) or completely automatic (in the case of a self-tuning tool, which implements the decision directly, without the intervention of the DBA).

However, DB tuning heuristics are typically empirically defined, and most lack a precise definition. For example, when the DBA decides on performing a tuning action (*e.g.,* rewriting a query) to improve the database performance, s/he is following reasoning that is present in her/his mind, based on concepts, principles, and previous experiences that only s/he is aware of. This scenario worsens in the presence of self-tuning tools when the DBA is unaware of the rationale followed by the tool since only the final physical design of the tuning process is available. The DBA cannot assess the tuning actions or even backtrack an individual step taken by the tool. For example, the DBA can conclude that a particular index created by the self-tuning tool might degrade the performance of insertion operations in the DB and then may want to undo a tuning action executed by the heuristic. However, the heuristic might have considered that, even while degrading insert operations, the index was still beneficial to the database workload as a whole (the set of all queries and data manipulation commands), as a result of a decision rationale unknown to the DBA. In this case, the rationale and the concepts analyzed by the tool are embedded in its source code, thus making it difficult or even impossible for the DBA to scrutinize.

Also, semi-automatic or self-tuning tools are tailored to suggesting specific actions for particular DBMSs, which makes heuristics experiments difficult in scenarios including several DBMSs and their extension to different structures and parameters. This difficulty does not only occur because the user must understand the source code of the tool, but also because each DBMS employs distinct concepts and terminology for its physical data structures and parameters, as well as for its implementation and syntax.

In this paper, we propose an approach to minimize these problems by explicitly representing the elements and actions involved in the database tuning heuristics using ontologies, as they represent an adequate means to support an accurate semantic record of the heuristics formulation and behavior. The DBA can analyze the tuning heuristics in more detail, as well as compare the different alternatives that each possible heuristic proposes. Also, the DBA can perform (semi-) automatic experiments by combining distinct heuristics. Finally, the use of ontologies also increases the heuristics understanding of the DBA, since they are described using higher-level concepts.

Thus, we present an ontological perspective for DB tuning heuristics. Our approach is developed on top of an ontology-driven conceptual framework[1], which includes an ontology of DB tuning heuristics and an ontology of DB tuning structural concepts.

[1] https://github.com/BioBD/outer_tuning, last accessed 2019/04/07.

These ontologies may be extended straightforwardly at the conceptual level to include new concepts and heuristics, without requiring the understanding of the implementation code. The advantage of our approach is that it provides a transparent methodology for tuning databases using multiple heuristics defined through rules based on semantic reasoning. One may create new heuristics by reasoning over the ontologies. We discuss some results of the application of our framework to two real-world case studies.

The remainder of this paper is organized as follows. In Sect. 2 we discuss state of the art in this area; in Sect. 3 we provide an overview of our conceptual framework, and we detail the ontology responsible for defining and executing the heuristics; Sect. 4, we showcase an outline of a demonstration of our framework, using our heuristic ontology. Section 5 concludes this research work.

2 State-of-the-Art

To set the context for discussing the specific contributions of this work, it is important to present different types of initiatives that consider ways of capturing and representing concepts involved in the tuning process, as well as those which offer any tuning support based on the represented concepts for DB or DB Tuning heuristics. As the proposition of specific DB tuning heuristics is not the focus of our proposal, we refrain from citing here such related literature.

The first group of works contemplates those related to conceptual models associated with the DB domain. The Common Warehouse Metamodel (CWM) specification, proposed by the Object Management Group (OMG) as a metadata interchange standard [2] comprises, among others, a package to describe relational data resources, which includes elements associated to indexes, primary keys, and foreign keys. Also supporting interoperability, Aguiar, and colleagues [3] proposed RDBS-O, a well-founded reference ontology covering high-level DB structure concepts. Both initiatives serve as a starting point for more comprehensive efforts covering other aspects of DB physical design, optimization and tuning. Aligned with our work, Ouared and colleagues adopt an approach that proposes a meta-advisor repository for DB physical design [4]. They describe a metamodel with elements dedicated to express optimization algorithms (heuristics), their characteristics and a cost model.

The second group of initiatives addresses tools [5–10] and approaches [11–13], which help the DBA to improve the performance of DBMSs. Considering the use of previously represented knowledge, the research work of Bellatreche and colleagues [14, 15] tracks the relationship between the requirements of the application and the suggestion of physical structures for optimization of the DB in a data warehouse environment. It uses an ontology to formalize domain concepts from data sources and SWRL rules [16] to relate them to the application requirements. Although the strategy supports indexes suggestions, there is no further explanation for them or the proposal of grounded alternatives. Recently, Zhang and colleagues presented the OtterTune tool, which leverages data collected from previous tuning efforts to train machine learning models, and recommends new configurations that are as good as or better than ones generated by existing tools or a human expert [17]. Among the helpful suggestions, they highlight the selection of the best access structures and configuration parameters for the DBMS. Zhang and colleagues proposed the CDBTune, an end-to-end automatic DBMS configuration tuning system that recommends superior knob settings in cloud environments, using deep reinforcement learning (RL). Through the reward-feedback mechanism in RL instead of traditional regression, they expect to accelerate the convergence speed of their model and improve efficiency of online tuning [18]. Zheng and colleagues presented a neural network-based algorithm for performance self-tuning [19] based on the workload and identify key system performance parameters, suggesting values to tune the DBMS. The reasoning of the algorithms proposed by these works is hidden in the source code or in the constructed model, making it difficult for the DBA to understand it and make adjustments to extend, add or combine heuristics. The decisions are not explained or justified to the DBA. In certain systems, such as Oracle DBMS, there are specific tools for suggesting tuning actions, also making the rationale of their choices available [20, 21]. Nevertheless, they still fail to capture the actual DBA decision process, with justifications for chosen and refused suggested tuning alternatives. Although there are a variety of tools and strategies to support the DBA in the DB tuning task, most works fail to make reasoning and decisions explicit.

Moreover, existing DB tuning tools and approaches suggest actions but do not provide a higher level mechanism for the DBA to verify the effectiveness of their actions and to tailor the rationale to his/her background knowledge. If the DBA disagrees with something suggested by the tool or approach, it may not be feasible for the DBA to include or change any reasoning proposed by the tool. That way, only an experienced DBA would be able to select the best tool to assist him/her according to the established scenario. Besides, the tools and approaches that involve more than one technique or structure do not have enough flexibility to consider additional methods.

3 The Outer-Tuning Conceptual Framework

We have developed Outer-Tuning, an ontology-driven framework for DB Tuning. Outer-Tuning works both in automatic (self-tuning) and in semi-automatic (human intervention) modes, and it may be extended to address new data structures and heuristics because all changes are expressed in the conceptual level using a declarative language.

Outer-Tuning allows the user to enable/disable a set of inference rules (SWRL rules [16]) established by heuristics. This functionality is crucial for the DBA, as s/he can choose, through the interface, which heuristics should be used to tune the DB. The use of a declarative language to define heuristics allows the DBA to focus on "what" the heuristic should do, instead of "how" it should work.

3.1 Outer-Tuning Overview

Figure 1 Illustrates the architecture and execution of the Outer-Tuning framework, using the activity diagram (UML alternative to the BPMN Business Process diagram)[2].

[2] https://sparxsystems.com/enterprise_architect_user_guide/14.0/guidebooks/tools_ba_uml_activity_diagram.html, last accessed 2019/04/07.

Outer-Tuning monitors the DB [Step 1] in a non-intrusive and continuous way and captures the workload submitted to the DB. Then, queries are parsed to identify query components and data structures, whose corresponding concepts in the ontology are needed by the tuning heuristics to estimate their processing cost (preconditions) [Step 2]. The rules engine instantiates the concepts [Step 3] and applies the tuning heuristics specified in the ontology rules, thus inferring tuning actions [Step 4] to improve the performance of the queries in the workload.

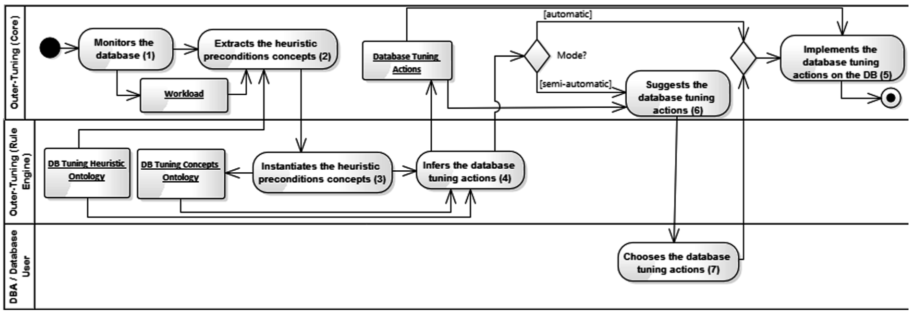
Outer-Tuning (Core)
Outer-Tuning (Rule Engine)
DBA / Database User
Monitors the database (1)
Extracts the heuristic preconditions concepts (2)
Workload
Database Tuning Actions
[automatic]
Mode?
[semi-automatic]
Suggests the database tuning actions (6)
Implements the database tuning actions on the DB (5)
DB Tuning Heuristic Ontology
DB Tuning Concepts Ontology
Instantiates the heuristic preconditions concepts (3)
Infers the database tuning actions (4)
Chooses the database tuning actions (7)

Fig. 1. Outer-tuning execution process

When the framework is set to automatic mode [Step 5], all suggested tuning actions (e.g., creation of indexes and materialized views) are applied to the DB; in semi-automatic mode, the suggested tuning actions are presented to the user [Step 6], who may customize [Step 7] which tuning actions will actually be applied to the DB [Step 5].

3.2 Conceptual Models

The structural conceptual model of the Outer-Tuning framework comprises two independent and complementary (sub)ontologies[3]: the DB tuning concepts ontology (namespace: *tuning*) and the DB tuning heuristic ontology (namespace: *heuristic*). The DB tuning heuristic ontology, which is the focus of the present work, makes references to the concepts described in the DB tuning concepts ontology.

The ontologies are capable of answering the following competency questions: (i) which are the DB concepts involved in the DB tuning process? (ii) what is necessary for each heuristic to know about the DB and to make its decisions? (iii) which are the possible actions that can be performed by a heuristic?

In this paper, we focus on the DB tuning heuristic ontology, which addresses questions (ii) and (iii), and relies on the Unified Foundational Ontology (UFO) [22, 23] as its semantic foundation. Using UFO allows removing the ambiguity of the concepts used in the task of running DB tuning heuristics. This article explains the process of development and use of the ontology.

[3] https://www.ime.uerj.br/ontuning/, last accessed 2019/04/07.

The heuristic ontology was designed to support the specification of DB tuning heuristics as rules and the dynamic execution of heuristics during DB tuning, as explained in the following Subsections.

Heuristic Specification

The DBA (which is the role played by a *Person* while tuning a DB) chooses and specifies the heuristics (Fig. 2) that should be considered for DB tuning. Throughout the tuning task, the DBA assumes more specific roles with distinct goals. For example, when defining (and configuring) a heuristic the *heuristic:DBA* plays the role of a *heuristic:specifier*. When configuring a heuristic, a *heuristic:HeuristicSpecification* relationship arises. The *heuristic:Heuristic* exists independently of other concepts.

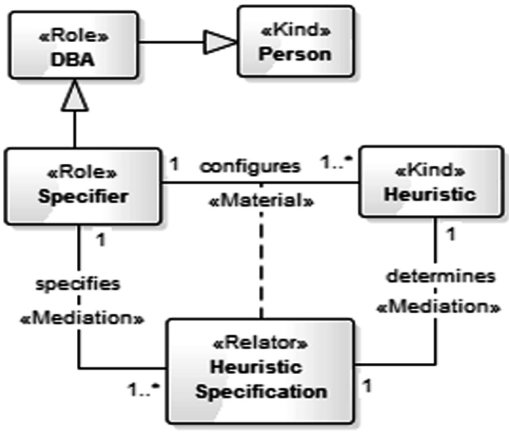

Fig. 2. Fragment of the heuristic ontology – heuristic specification

A *heuristic:Heuristic* takes data structures of the DB schema (defined in DB tuning concepts ontology) and cost estimates of DB operations into account to infer those actions that we should execute for performance improvement. Thus, each precondition of a *heuristic:Heuristic* references existing database concepts (*heuristic:DBConcept*) (Fig. 3). The *heuristic:PreconditionConcept* is derived from the relationship between *heuristic:Heuristic* and *heuristic:Database Concept*. A *heuristic:DBConcept* specializes into *heuristic:TuningAssist* or *heuristic:DBObject*. The *heuristic:TuningAssist* is a database concept required by a heuristic to perform its suggested actions (for example, a heuristic that suggests indexes needs to execute the "create index" DDL statement. So, this heuristic requires the "create index" statement as a precondition). A *heuristic: DBObject* refers to the database concept analyzed by the heuristic to make decisions (for example, the same heuristic suggesting an index may need to examine the *tuning: Where* clause of an SQL statement. So, the *tuning:Where* clause concept is a precondition of this heuristic).

Each *heuristic:DBConcept* defined in the DB tuning concepts ontology needs to be instantiated (Fig. 4) by a *heuristic:Source* (either a *heuristic:Function* or a *heuristic: Rule*). A *heuristic:Function* is a *heuristic:ConceptInstanceFunction* when it instantiates a *heuristic:DBConcept* (that is, when it creates a new individual in the DB tuning concepts ontology). For example, a *heuristic:ConceptInstanceFunction* captures the database workload and instantiates a *heuristic:DMLcommand* concept with the specific query command captured. Analogously, a *heuristic:Rule* is a *heuristic:ConceptInstanceRule* when it instantiates a *heuristic:DBConcept*.

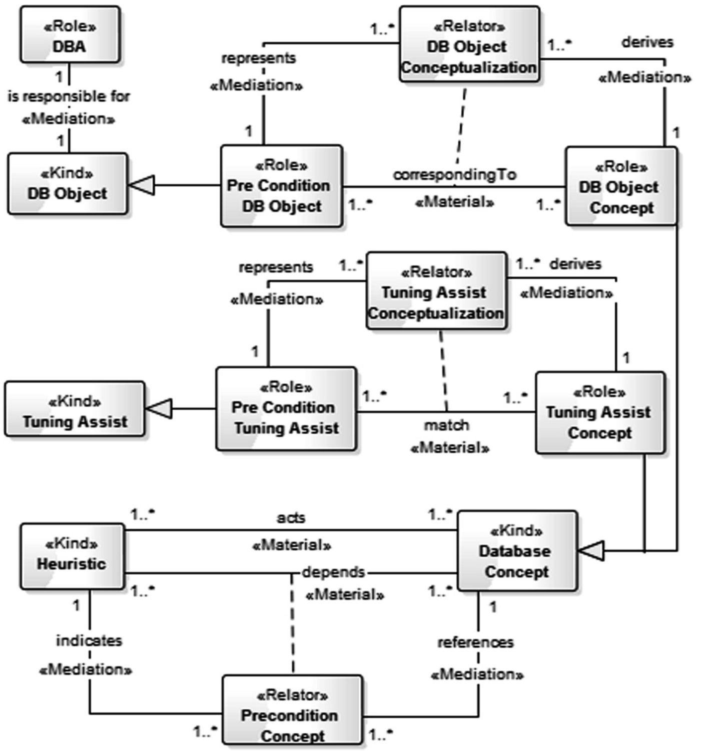

Fig. 3. A fragment of the heuristic ontology – heuristic preconditions definition

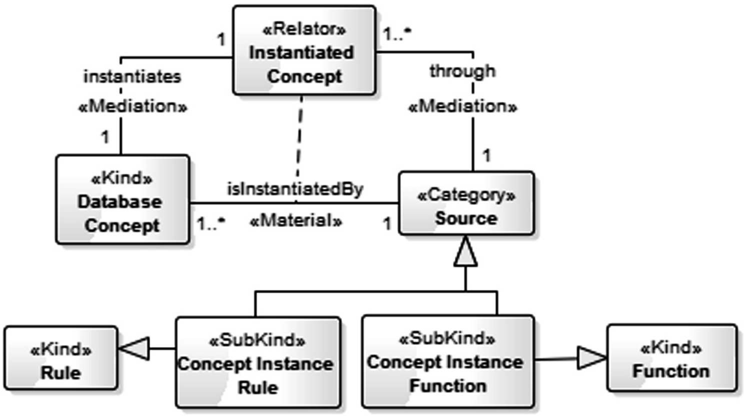
«Relator» Instantiated Concept
instantiates «Mediation»
through «Mediation»
«Kind» Database Concept
isInstantiatedBy «Material»
«Category» Source
«Kind» Rule
«SubKind» Concept Instance Rule
«SubKind» Concept Instance Function
«Kind» Function
1
1..*

Fig. 4. Fragment of the heuristic ontology – heuristic concepts instantiation definition

Functions differ from rules since a *heuristic:Function* can interact with the DBMS, whereas a *heuristic:Rule* cannot. This is important since some *heuristic:DBConcept*s require accessing the database content to be instantiated, such as DML (Data Manipulation Language) commands. For example, the *tuning:DMLcommand* concept, defined in the domain ontology, requires a function that connects to the DBMS to retrieve the *tuning:DMLcommand* from the workload submitted by the user.

A *heuristic:Function* or method (Fig. 5) is a collection of statements embedded in the database that operate together in a group. It defines input and output parameters. In theory, all these parameters are optional. In our approach, the output parameter (instantiated concept or tuning action) is mandatory. For example, a *heuristic:Function* that retrieves a *tuning:DMLcommand* from the DBMS does not need any input parameters, while a *heuristic:Function* that returns information from an execution plan requires the execution plan as its input parameter. The *heuristic:Parameter* is a concept in the ontology, and its properties indicate if it plays the role of an input or of an output parameter to a *heuristic:Function*. Our approach preferably instantiates concepts through rules rather than functions, so as to make any reasoning explicit. Functions are used in specific situations to instantiate concepts that cannot be derived by rules.

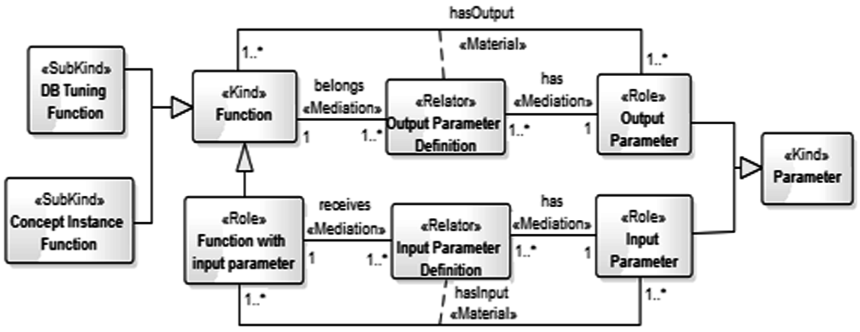

Fig. 5. Fragment of the heuristic ontology – functions definition

A *heuristic:Rule* (Fig. 6) is composed by a *heuristic: condition-action* pair, meaning that when the condition is satisfied, an action is performed. They provide the DBA with the rationale for heuristics application, by keeping track of the way concepts are instantiated when each tuning alternative is evaluated.

A *heuristic:RuleEngine* represents a system that applies inference mechanisms based on given rules, which in our case define DB tuning heuristics. The rule engine component implements the code that selects and executes the heuristics (described in the ontology) to suggest tuning actions (e.g., query rewriting) and the creation of access structures (e.g., materialized views). The instance of *tuning:DMLcommand* comprises the where clauses according to the SQL specification.

To perform the DB tuning task, *heuristic:Heuristic* defines *heuristic:Rules* that must be evaluated by the inference engine. Unlike the *heuristic:ConceptInstanceRule*, this kind of rule (*heuristic:HeuristicDefinitionRule*) corresponds to the definition of the heuristic's actions about the DB behavior. For example, a heuristic of indexes can simulate indexes, called hypothetical, to know if the optimizer would use it or not. A *tuning:HypotheticalIndex* could exist in the DB (not physically) to improve certain *tuning:DMLcommand* performance.

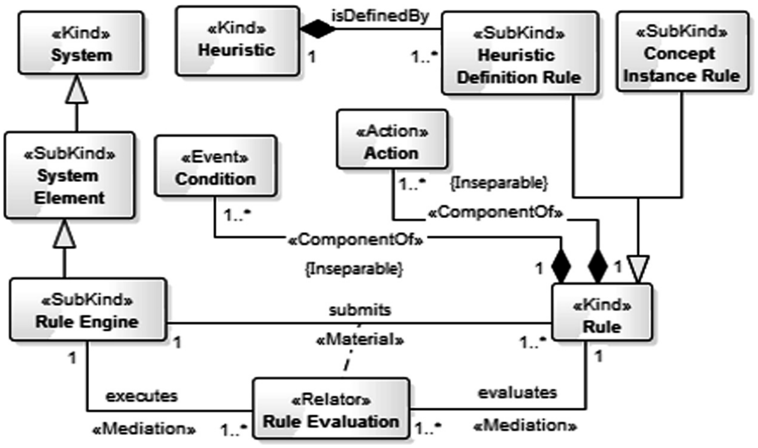

Fig. 6. Fragment of the heuristic ontology – rules definition

The heuristic may only suggests a hypothetical index, as illustrated by the following rule:

```
SingleStatement (?stat) ^ Where (?where) ^ hasClause (?stat, ?where)
^ Predicate (?pred)
^ componentOf (?pred, ?where) ^ SimpleExpression (?pred) ^ Refer-
encedColumn (?refCol)
^ componentOf (?refCol, ?pred) ^ hasName (?refCol, ?nameCol)
^ swrlx:makeOWLThing(?hypInd, ?refCol) ^ swrlb:stringConcat (?name-
Hyp, "hi_", ?nameCol)
->  HypotheticalIndex (?hypInd) ^ hasName (?hypInd, ?nameHyp) ^
originates (?stat, ?hypInd)
^ acts_on (?hypInd, ?refCol) ^ IndexedColumn (?refCol)
```

This rule analyzes the *tuning:SingleStatement* and suggests the indexes according to the columns referenced in the *tuning:Where* clause. Given a *tuning:SingleStatement*, if it has a *tuning:Where* clause predicate that is a simple expression, the rule engine should get the referenced column and its name. As a consequence (the rule action), the machine creates a new individual of type *tuning:HypotheticalIndex* with its relationships with the *tuning:SingleStatement* that caused it (Originates) and the *tuning:Column* being indexed (Acts On). The *tuning:IndexedColumn* is the same as the *tuning:ReferencedColumn*. Additional rules defined in the ontology[4] were not mentioned here, since we described only the ones relevant to the example in the article.

[4] http://www.inf.puc-rio.br/~postgresql/conteudo/projeto4/download/OntologiaTuning.owl, last accessed 2019/04/07.

All the concepts referring to data structured of the DB schema are defined in the DB tuning concepts ontology, to reflect the structures considered by common DB Tuning heuristics [7, 24, 25], and includes *tuning:DMcommand*, *tuning:DDLcommand*, *tuning:Clause*, *tuning:AccessStructure*, *tuning:Index*, *tuning:RealIndex*, *tuning:HypotheticalIndex*, *tuning:Table*, *tuning:Column*. If a new DB tuning heuristic needs to refer to a new concept not previously defined, it just needs to extend the DB tuning concepts ontology, defining the semantics of this new concept according to the existing ones.

Execution of Heuristics

The subsequent moment in the DB tuning task is the heuristic execution. The *heuristic:HeuristicExecution* needs the *heuristic:HeuristicDefinition* described in the heuristic ontology since it is responsible for suggesting or applying tuning techniques based on the workload analysis. The user starting the execution process does not have to be an expert, since the heuristics have been specified by an expert (DBA). When a person starts a heuristic execution, s/he chooses which heuristic s/he wants to execute first, assuming the *heuristic:Executor* Role and performing the *heuristic:HeuristicSelection*.

At runtime, the preconditions concepts, defined previously, must be instantiated (*heuristic:InstantiatedConcepts*) and retrieved by a software agent. The software agent is defined since an automated self-tuning strategy (without human DBA intervention) may be used. Then, the *heuristic:Agent* manages the *heuristic:HeuristicExecution*, using *heuristic:InstantiatedConcepts*. During the *heuristic:HeuristicExecution*, each *heuristic:Rule* is checked and evaluated by the rule engine (*heuristic:RuleEvaluation*).

Rule consequents represent suggested Actions, which can be either a *heuristic:TuningAction* or a *heuristic:RuleAction*. When there is a *heuristic:RuleAction*, it becomes a *heuristic:VariablesControl* (*Relator*) managed by the *heuristic:RuleEngine*. We consider *heuristic:VariablesControl* as new instances/individuals of objects or properties in the domain ontology according to the conditions established by the heuristic. For example, a heuristic can add a bonus property to hypothetical indexes each time they are mentioned in an execution plan. This action (adding bonus) cannot be considered a tuning action because it is just a simulation. Therefore, we modeled it as a *heuristic:RuleAction*, as a logical consequence of *heuristic:VariablesControl* (bonus).

A *heuristic:TuningAction* is an action to improve the DB performance, such as the creation of an access structure. To illustrate it, the heuristic may decide to transform the hypothetical index into a real index when it achieves a considerable bonus. Every *heuristic:TuningAction* requires an interaction with the DB, that is achieved via a *heuristic:Function* to perform this interaction, represented as a *heuristic:DBTuningFunction*.

Moreover, every Tuning Action may have an optional Justification, so roles were created (*heuristic:Justifier* and *heuristic:TuningActionJustifie*r) to contemplate it. The justification indicates an explanation for the fact that the DBA accepted or not an action.

The DB tuning task can be semi-automatic or fully automatic. To address this, we introduce the concept of *heuristic:Implementer* that aggregates properties from *heuristic:DBA* and *heuristic:Agent*. When the *DBA* implements the action (*DBA Implementer*), it means that it is semi-automatic, i.e., the DBA needs to intervene and indicate whether or not to accept a suggested action. The DBA will need to analyze all the suggested action. When the agent implements the action (*heuristic:AgentImplementer*), it means that it is fully automatic, i.e. without any human intervention, and the *heuristic:Agent* performs all of the suggested actions.

Whenever the heuristic ontology is instantiated it means that a given heuristic is being performed using concept instances defined in DB tuning concepts ontology according to the workload submitted to the DB.

4 Case Studies

We ran two scenarios in our Outer-tuning framework, using our ontologies. The first scenario applied tuning heuristics that suggest (hypothetical and physical) index structures, and shows the importance of visualizing all the alternatives considered by the tuning heuristics, instead of only the implemented actions. The second scenario illustrates the need of extending the ontologies used by the framework, adding tuning heuristics that suggest materialized views. Ontology extension points are described in [26].

Scenario 1. The DB has a table (EMPLOYEE) with four columns (Identifier, Name, Gender and Salary). Figure 7 shows the query presented in the DB workload that was analyzed by the tuning heuristics defined in the heuristic ontology.

```
SELECT * FROM EMPLOYEE
WHERE salary in (1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000)
AND  gender = “M”’;
```

Fig. 7. Query presented in the analyzed workload

The DBA selects two different heuristics (HEIC-A and HEIC-B) to run individually on the same workload and DB. The two heuristics eventually recommended the creation of different indexes in the DB.

At first, the DBA executes the framework using only the rules of the HEIC-A heuristic about hypothetical indexes. Analyzing the DB, the optimizer generates the query execution plan that chooses to use two hypothetical indexes (over the salary column and gender column) created by the HEIC-A heuristic. Later, on the same initial state of the DB (i.e., with no tuning actions performed by HEIC-A), the DBA executes the framework using only the rules of the HEIC-B heuristic. The optimizer generates the query execution plan that does not use any index and, rather, suggests a full table scan operation. From these outcomes, the DBA may be in doubt about which recommendation to follow, either creating the physical index or not. The DBA, then, checks the rule conditions that generated the candidate indexes and identifies that both heuristics consider the columns referenced in the WHERE clause. As the conditions of the rule are the same, the DBA cannot understand why the heuristics generated different recommendations. S/he verifies information about the candidate index suggested by each heuristic. By analyzing the ontology instances created during the execution of

Scenario 1, the DBA concludes that HEIC-B heuristic does not consider bitmap indexes, while HEIC-A does (Table 1). The following SQWRL query [27] recovers all hypothetical indexes suggested per heuristic:

```
Heuristic(?h) ^ DatabaseConcept(?dc) ^ acts (?h, ?dc) ^
TuningAssist(?tunAss)
^ match(?tuna, ?dc) ^ CreateHypothetical(?tunAss) ^
HypotheticalIndex(?hypInd)
^ creates(?tunAss, ?hypInd) ^ hasType(?hypInd, ?indType)
-> sqwrl:select(?h, ?indType)
```

The regular B+tree indexes created by HEIC-B leads to a higher execution cost to the query, justifying the fact that the optimizer does not consider them. The DBA may also assess the decision rationale followed by our proposal (i.e., all alternative considerations, even for those that are not considered for suggesting tuning actions).

Table 1. Hypothetical Indexes considered by heuristics HEIC-A and HEIC-B

Heuristic	Index		
	Name	Creation cost	Type
HEIC-A	HI_GENDER	0.5625	Bitmap
HEIC-A	HI_SALARY	4	Bitmap
HEIC-B	HI_GENDER	13	B+tree
HEIC-B	HI_SALARY	17	B+tree

Scenario 2. Scenario 2 used the TPC-H benchmark for the workload. This benchmark has analytical queries (OLAP), an opportunity for the evaluation of heuristics that consider materialized views (MVs). We considered two types of MVs during the analysis of the results: the beneficial and the malefic. A MV is considered *beneficial* when rewritten queries that use the MV increases the performance of the workload, and *malefic* when their use decreases the performance of the workload. This occurs when the MV size is greater than the number of pages read in the original query.

We ran two additional heuristics [28] in our Framework, which have inferred materialized views for the workload. HMV1 pointed out three beneficial materialized views to the workload (Q01, Q05, Q09) and two malefic (Q04, Q12). HMV2 showed four beneficial materialized views (Q01, Q05, Q06, Q09) and five malefic (Q03, Q04, Q07, Q12, Q14). Both heuristics brought a positive gain equivalent to the workload (12.4% and 12.2%). But the framework showed that HMV1 estimated lower costs than HMV2 for the creation and storage of MVs. While HMV1 proposed the creation of 5 MVs, HMV2 proposed 9. This demonstrates the way the framework shows that the same heuristic can bring benefits or losses depending on the workload received.

Regardless of the result presented by both tuning heuristics, this scenario shows that the framework is able to work simultaneously with more than one heuristic, compare the solutions presented and the inclusion of new heuristics. With the use of our framework, the DBA has sufficient information to assess which heuristics are interesting for his/her workload, in a conceptual level. An experienced DBA may even extend the heuristics behavior based on his/her experience. For example, composing HMV1 and HMV2 would suggest the beneficial MVs both have inferred and avoid the malefic MVs (Q7, Q3, Q14) suggested by HMV2.

In conclusion, the scenarios show that the Framework is able to: (i) Display all alternatives evaluated, regardless of the decision that the heuristic took. In addition to the alternatives, the DBA may also submit SQWRL queries to view the instances of the ontology that represent all the behavior of the heuristics; (ii) infer useful DB tuning actions. The case studies have demonstrated useful actions to improve database performance; (iii) compare tuning heuristics described with the ontology. The case studies show that the comparison of heuristics is possible through the interpretation of ontology instances, and (iv) support the DBA in the DB tuning task with relevant information which s/he can match the heuristics to the workload or insert new heuristics in the tuning ontology.

5 Conclusion and Future Works

We presented an ontological perspective for DB tuning heuristics execution. Existing proposals hinder both the extension to new heuristics and the transparency of the reasoning used for decision-making. We described a conceptual model (heuristic ontology) to address these points. The extension to new heuristics is facilitated because the user only needs to instantiate the heuristic ontology defining new rules. Although not all database tuning strategies are covered by our ontology, the DB tuning concepts ontology (not detailed in this present paper) can be easily extended (without changing the framework) to include concepts and properties for memory tuning (e.g.: shared_buffers, checkpoint_segments), query tuning (e.g.: rewriting) and transactions tuning (e.g.: locks, isolation levels). The transparency of the reasoning is obtained by defining the rationale described in rules. As the rules are defined in terms of familiar concepts, it becomes easier for the DBA to understand the heuristics proposed behavior. We argue that representing database tuning activities with more formalism through our ontologies allows a more precise discussion and study of the issues behind database tuning—for example, the different ways of working with tuning heuristics (add new structures to existing heuristics, work with different heuristics that suggest the same type of structure)—that are not possible if the concepts involved were not explained through a high-level model.

Future work will address new DB tuning heuristics and how we can compose or combine them, identifying conflicts of the rules (semi) automatically and solving them. Besides, we can create a repository for DB tuning decisions rationale.

References

1. Shasha, D., Bonnet, P.: Database Tuning: Principles, Experiments, and Troubleshooting Techniques. Morgan Kaufmann Publishers, San Francisco (2003)
2. OMG (Object Management Group): Common Warehouse Metamodel (CWM) Specification. Version 1.1, vol. 1, No. formal/03-03-02 (2003)

3. Aguiar, C.Z., Falbo, R.D., Souza, V.E.: Ontological representation of relational databases. In: ONTOBRAS (2018)
4. Ouared, A., Ouhammou, Y., Roukh, A.: A meta-advisor repository for database physical design. In: Bellatreche, L., Pastor, Ó., Almendros Jiménez, J., Aït-Ameur, Y. (eds.) MEDI 2016. LNCS, vol. 9893, pp. 72–87. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45547-1_6
5. Ding, Z., Wei, Z., Chen, H.: A software cybernetics approach to self-tuning performance of on-line transaction processing systems. J. Syst. Softw. **124**, 247–259 (2017)
6. Noon, N.N., Getta, J.R.: Automated performance tuning of data management systems with materializations and indices. J. Comput. Commun. **4**, 46–52 (2016)
7. Morelli, E., Almeida, A., Lifschitz, S., Monteiro, J.M., Machado, J.: Autonomous re-indexing. In: Proceedings of the ACM Symposium on Applied Computing (SAC), pp. 893–897 (2012)
8. Bruno, N., Chaudhuri, S., König, A.C., Narasayya, V., Ramamurthy, R., Syamala, M.: AutoAdmin project at Microsoft research: lessons learned. Bull. IEEE Comput. Soc. Tech. Comm. Data Eng. **34**(4), 12–19 (2011)
9. Rangaswamy, S., Shobha, G.: Online indexing for databases using query workloads. Int. J. Comput. Sci. Commun. **2**(2), 427–433 (2011)
10. Goasdoué, F., Karanasos, K., Leblay, J., Manolescu, I.: View selection in semantic web databases. Proc. VLDB Endow. **5**(2), 97–108 (2012)
11. Basu, D., et al.: Regularized cost-model oblivious database tuning with reinforcement learning. In: Hameurlain, A., Küng, J., Wagner, R., Chen, Q. (eds.) Transactions on Large-Scale Data- and Knowledge-Centered Systems XXVIII. LNCS, vol. 9940, pp. 96–132. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53455-7_5
12. Bellatreche, L., Schneider, M., Lorinquer, H., Mohania, M.: Bringing together partitioning, materialized views and indexes to optimize performance of relational data warehouses. In: Kambayashi, Y., Mohania, M., Wöß, W. (eds.) DaWaK 2004. LNCS, vol. 3181, pp. 15–25. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30076-2_2
13. Bouchakri, R., Bellatreche, L.: On simplifying integrated physical database design. In: Eder, J., Bielikova, M., Tjoa, A.M. (eds.) ADBIS 2011. LNCS, vol. 6909, pp. 333–346. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23737-9_24
14. Bellatreche, L., Khouri, S., Boukhari, I., Bouchakri, R.: Using ontologies and requirements for constructing and optimizing data warehouses. In: Proceedings of International Convention MIPRO, pp. 1568–1573 (2012)
15. Khouri, S., Bellatreche, L., Boukhari, I., Bouarar, S.: More investment in conceptual designers: think about it! In: Proceedings of the IEEE International Conference on Computational Science and Engineering, pp. 88–93 (2012)
16. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: a semantic web rule language combining OWL and RuleML. National Research Council of Canada, Network Inference, and Stanford University (2004)
17. Zhang, B., et al.: A demonstration of the ottertune automatic database management system tuning service. Proc. VLDB Endow. **11**(12), 1910–1913 (2018)
18. Zhang, J., et al.: An end-to-end automatic cloud database tuning system using deep reinforcement learning. In: Proceedings of the 2019 International Conference on Management of Data, pp. 415–432. ACM (2019)
19. Zheng, C., Ding, Z., Hu, J.: Self-tuning performance of database systems with neural network. In: Huang, D.-S., Bevilacqua, V., Premaratne, P. (eds.) ICIC 2014. LNCS, vol. 8588, pp. 1–12. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09333-8_1
20. Dias, K., Ramacher, M., Shaft, U., Venkataramani, V., Wood, G.: Automatic performance diagnosis and tuning in oracle. In: Proceedings of CIDR Conference, pp. 84–94 (2005)

21. Alhadi, N., Ahmad, K.: Query tuning in oracle database. J. Comput. Sci. **8**(11), 1889–1896 (2012)
22. Guizzardi, G.: Ontological foundations for structural conceptual models. Thesis presented in the University of Twente (2005). http://doc.utwente.nl/50826
23. Guizzardi, G., Wagner, G., de Almeida Falbo, R., Guizzardi, R.S.S., Almeida, J.P.A.: Towards ontological foundations for the conceptual modeling of events. In: Ng, W., Storey, V.C., Trujillo, J.C. (eds.) ER 2013. LNCS, vol. 8217, pp. 327–341. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41924-9_27
24. Lohman, G., Valentin, G., Zilio, D., Zuliani, M., Skelley, A.: DB2 advisor: an optimizer smart enough to recommend its own indexes. In: Proceedings of the IEEE International Conference on Data Engineering (ICDE), pp. 101–110 (2000)
25. Bruno, N.: Automated Physical Database Design and Tuning. CRC Press, Boca Raton (2011)
26. Oliveira, R.P., Baião, F., Almeida, A.C., Schwabe, D., Lifschitz, S.: Outer-tuning: an integration of rules, ontology and RDBMS. In: Proceedings of the Brazilian Symposium on Information Systems (SBSI), Aracaju, Sergipe, Brazil (2019)
27. O’Connor, M.J., Das, A.K.: SQWRL: a query language for OWL. In: Proceedings of the 5th International Workshop on OWL: Experiences and Directions, OWLED (2009)
28. Oliveira, R.P.: Ontology-based database tuning: the case of materialized views. Master thesis presented at PUC-Rio, Rio de Janeiro, Brazil (2015)