Outer-Tuning: an Ontology-based Extensible Framework for Supporting Database Automatic Tuning

Raphael Marins[1], Rafael Pereira de Oliveira[2], Edward Hermann Haeusler[2], Sérgio Lifschitz[2], Daniel Schwabe[2] and Ana Carolina Almeida[1]

[1]Department of Informatics and Computer Science - UERJ, Brazil
[2]Department of Informatics - PUC-Rio, Brazil
marins.raphael@graduacao.uerj.br, {rpoliveira, hermann, sergio, dschwabe}@inf.puc-rio.br, ana.almeida@ime.uerj.br

Abstract. This paper presents the Outer-Tuning framework, which aims to support the (semi) automatic tuning of relational database systems through a domain-specific ontology. Ontologies have shown themselves to be increasingly promising, adding semantics and standardizing the different terms used in a domain. Thereby, our framework seeks to explain and make explicit the tuning heuristics reasoning while enabling the evaluation of new ontology-inferred methods. In this paper we focus on the main aspects of the Outer-Tuning component-based architecture. We also give an overview of our tool in practice. Finally, we show two useful extensions, concerning new DBMSs and a way of dockerizing into a container.

Categories and Subject Descriptors: H.2.2 [**Physical Design**]: Access methods

Keywords: database tuning, docker, ontology, semantic framework

1. INTRODUCTION

Database administration and tuning are complex tasks that require specialization and fundamental knowledge in this area of computing. Several tools were developed to support the DBA (database administrator) in these activities. Some tools allow database (DB) tuning tasks to be performed (semi) automatically [Shasha and Bonnet 2002][Bruno 2011]. The DB tuning seeks for better performances of database systems: greater efficiency or transaction throughput. Adjustments are made to parameters, physical design, selection of access structures, always according to the DB workload.

In DB tuning tools, there is a lack of clarity about the decisions and actions that are taken automatically, making it difficult for the DBA to accept them blindly. Thus, Outer-Tuning proposes the use of a domain ontology for DB tuning (automatic or not) that provides a formal and explicit approach to decisions and inferences. The innovative contribution of this approach is to offer transparency and reliability about the alternatives available for possible scenarios in the Database Management System (DBMS), through concrete justifications for the decisions that were semantically defined.

This article is an extension of the paper presented by Almeida *et al.* (2018) [Almeida et al. 2018]. We describe the software architecture, functional and practical aspects of the Outer-Tuning framework, an ontology-based tool designed to support DBAs and developers in general in the decision-making involved DB tuning task. This tool aims to support the (semi) automatic DB tuning in relational DB systems. The tool's name originated because it presents all the alternatives analyzed by the heuristics and not only the one considered the best DB tuning action. An analogy is made to the relational

This work was partially supported by the Rio de Janeiro Research Foundation (FAPERJ), CNPq and CAPES Institutional Funding.
Copyright©2021 Permission to copy without fee all or part of the material printed in JIDM is granted provided that the copies are not made or distributed for commercial advantage, and that notice is given that copying is by permission of the Sociedade Brasileira de Computação.

operation of an outer join, which returns all tuples of the tables involved and not just those that satisfy the join condition. The basic idea of Outer-Tuning is to offer mechanisms that allow the DBA to make more consistent decision-making regarding possible tuning actions based on multiple alternatives and respective costs.

We next describe directly related work discussed in the literature. Section 3 presents an overview of the Outer-tuning architecture and Section 4 details framework extensions such as a new DBMS (*e.g.* MySQL) and the use of docker containers to facilitate the installation and use of our framework. Finally, section 5 concludes this work.

2. RELATED WORK

Zhang *et al.* presented the OtterTune tool [Zhang et al. 2018]. OtterTune collects knobs and metric data from past tuning sessions to train machine learning models (Gaussian Process Regression model). This model predicts how well the DBMS will perform with each possible configuration of DBMS. Thus, OtterTune optimizes the following configuration, gathering information to improve the model. The demonstration of OtterTune is carried out on PostgreSQL DBMS 9.6 with OLTP and OLAP workloads (TPC-C and TPC-H, respectively). Combining supervised and unsupervised machine learning methods, the authors proved that the tool could incorporate new information after observing the DB environment's behavior and the models learned from them, providing an increase in the efficacy of its recommendations. It is concluded that OtterTune was able to select impacting knobs in performance and recommend suitable knob configurations.

Zhang *et al.* proposed the CDBTune [Zhang et al. 2019]. This end-to-end automatic DBMS configuration tuning system recommends superior knob settings in cloud environments, using a try-and-error manner in deep reinforcement learning (RL). They expect to improve tuning efficiency. Experimental results showed that CDBTune improved performance with higher throughput and lower latency than other tuning tools through superior configurations.

Goasdoué *et al.* devise new algorithms for recommending view sets to be materialized (Materialized Views - MVs) according to the workload composed of RDF (Resource Description Framework) triples and rewriting queries based on RDF Schema [Goasdoué et al. 2012]. The proposed approach was implemented as a Java 6 application and used PostgreSQL 8.4.3. The experiments showed that the tool achieved cost reduction factors in many cases and recommended views that reduced query evaluation times by several orders of magnitude.

Some DBMSs have specific tools for suggesting tuning actions. Oracle DBMS has an Automatic Database Diagnostic Monitor (ADDM) tool that diagnoses bottlenecks and provides actionable recommendations to alleviate them [Dias et al. 2005][Alhadi and Ahmad 2012]. The experiments used Oracle 10g and proved that ADDM identified and correctly diagnosed several performance issues on internal production systems. Also, this tool makes the rationale of their choices available.

In Table I, we summarize and compare related work. Only our tool provides two ways to apply tuning actions. Although our tool does not yet have a defined heuristic in the ontology that adjusts the knobs, it may be extended to include this feature. Our tool is the only one that is concerned with adding semantics to the DB tuning process, thus allowing an explanation of the reasoning closer to the terms used by the DBA. In addition, this facilitates the extension of the framework, since the heuristics are also defined using the concepts known to the DB community. Outer-Tuning can be used with three DBMSs but has not yet been tested on cloud databases. Some tools (*e.g.*, ADDM) are already concerned with explaining the reasoning for their recommendations but end up failing in other aspects, such as the lack of flexibility in adapting to the different terms used by DBMSs and facilitating their extension for use in other DBMSs.

In general, none of the tools are not concerned with recording the DBA's justifications and his/her

Table I. Comparison of related work

Tool	Tuning action	Tuning target	Tuning semantics	DBMS support	Transparency in tuning suggestion (reasoning)	Final decision justification
OtterTune [Zhang et al. 2018]	automatic	knobs	No	PostgreSQL MySQL	No	No
CDBTune [Zhang et al. 2019]	automatic	knobs	No	Cloud databases	No	No
[Goasdoué et al. 2012]	manual	MVs	No	PostgreSQL	No	No
ADDM [Dias et al. 2005] [Alhadi and Ahmad 2012]	semi-automatic	Knobs Indexes	No	Oracle	Yes	No
Outer-Tuning	automatic semi-automatic	Indexes MVs	Yes	PostgreSQL Oracle, MySQL	Yes	Yes

tuning decisions. In a collaborative work (DBA team), it is important to record the history of decisions and their respective justifications. If a DBA A refuses a tuning action at first, it is important to record the reason for this refusal so that another DBA B, from the same team, will be aware when this suggestion comes back and consider DBA A's opinion.

3. OUTER-TUNING

We have developed Outer-Tuning, an ontology-driven framework for DB tuning, which works in automatic (self-tuning) and semi-automatic (human intervention) modes. Tuning heuristics are described as inference rules using the Semantic Web Rule Language (SWRL) [Horrocks et al. 2004] defined over an ontology. This set of inference rules may be enabled/disabled according to the user's preferences on the choice of the heuristic(s) that one wants to use to tune the DB. The video available at `http://www.inf.puc-rio.br/~postgresql/conteudo/projeto4/video/outertuning.mp4` shows a possible use of the tool.

3.1 Outer-Tuning Architecture

The architecture chosen for Outer-Tuning (Figure 1) is based on components. Given the tool's experimental features and the multiple technologies involved (*e.g.*, DBMSs, rules engine, ontology, libraries), it was decided that modularizing via components would facilitate communication among the technologies and leave each of the execution phases independent. Thus, if necessary, the components may be replaced (or maintained) individually.

As a result of choosing a component-based architecture, it was decided that Outer-Tuning would be developed as a framework application, which by definition, is a semi-complete application built with an organized collection of reusable software components [de Oliveira et al. 2011]. The choice of this type of framework with the use of components was made to enable future evolution towards service-oriented software with low coupling between the parts of the software.

Outer-Tuning components were defined and specialized for each cycle stage of its execution flow. Figure 1 lists the main elements of the proposed architecture, briefly described below:

(1) Database: any DB managed by a DBMS has its communication with the framework (6) and (12) through the connection drivers used by the components: WorkloadCollector (7) and TuningActionExecutor (11).

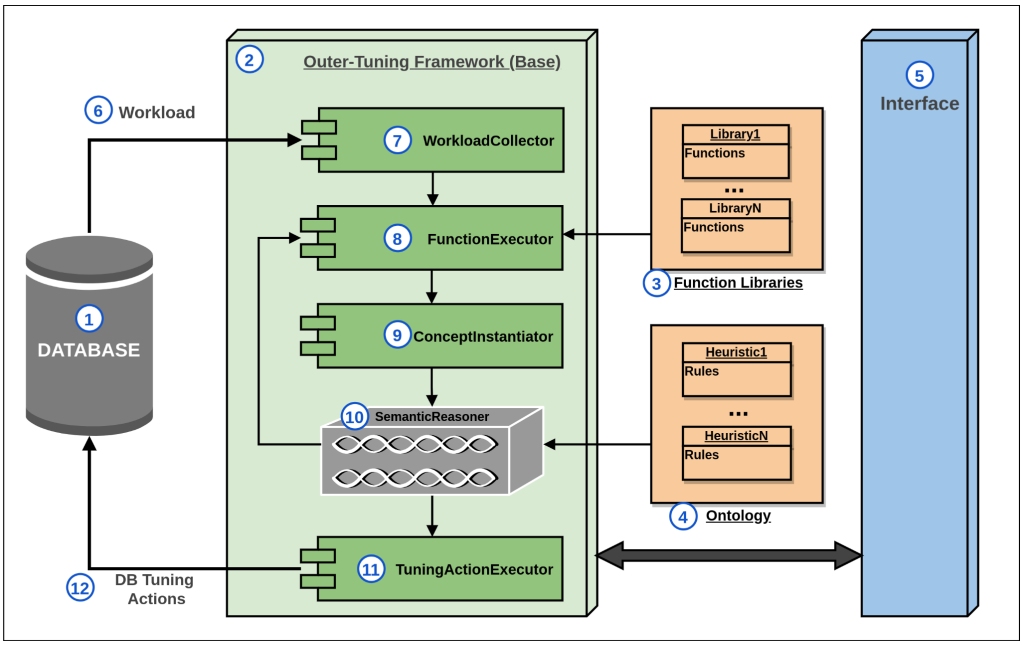

Fig. 1. Outer-Tuning Architecture

(2) Outer-Tuning Framework (Base): library of ordinary and redundant functions shared by components which may access the shared log.

(3) FunctionLibraries: gathers compiled source code libraries, responsible for extracting concepts from the workload. The libraries are read and executed at runtime, without intervention in the framework source code. There is an interface defined in FunctionExecutor (8) that searches in the repository for the desired functions (3).

(4) Ontology: contains concepts instantiated by the workload and heuristics through "if-then" rules). It is the main extensible part of the framework. The rules are defined in a declarative language (SWRL). We described part of the ontology (DB Tuning Heuristic Ontology) used by Outer-Tuning in [Almeida et al. 2019]. The ontology itself is available at `https://www.ime.uerj.br/ondbtuning/`.

(5) Interface: is responsible for interacting with the DBA. The communication with the framework (2) uses the blackboard design pattern [Khosla and Ichalkaranje 2005] due to its ease of implementation. We decided by an asynchronous approach. Mainly because it helps to scale the amount of queries to be captured and processed. The BlackBoard framework coordinates messages and designs a communication process able to extract data from queries incrementally. Thus, possible to be scaled by running multiple FunctionExecutors (8) at the same time.

(6) Workload: the captured workload consisting of SQL statements of the DML (Data Manipulation Language) type, their respective execution plans, and their frequency. This workload is automatically captured when the user submits statements to the database (1).

(7) WorkloadCollector: is the component responsible for collecting the workload with a time interval predetermined by the DBA to perform DB tuning through the JDBC driver.

(8) FunctionExecutor: extracts information from the workload and generates the individuals (instances) of the ontology (4) concepts, which are evaluated in the preconditions for the heuristics, and which must be instantiated and received from the semantic reasoner (10) to infer the tuning actions (12). The functions are searched in the FunctionLibraries (3). For example, the execution of a function that submits a query to DB metadata to get a table description (table name and its

columns). It was implemented using the Factory method pattern, one of the "Gang of Four" [Gamma et al. 1995] design patterns that describe how to solve recurring design problems to design flexible and reusable object-oriented software, that is, objects that are easier to implement, change, test, and reuse.

(9) ConceptInstantiator: instantiates the individuals of preconditions generated by the execution functions of FunctionExecutor (8) in the ontology (4).

(10) SemanticReasoner: is the component through which the rules defined in the ontology (4) are selected and executed. The Jess[1] engine was used for its implementation. There are inference rules for instantiation and tuning actions. For instantiation, the Semantic Reasoner receives and executes rules defined in the ontology (4). For tuning actions, it gets the instantiated concepts from the ConceptInstantiator (9) to infer new database tuning actions. Also, this reasoner sends the signature of the functions defined in the ontology (4) that need to be called by the FunctionExecutor (8) to instantiate concepts that cannot be instantiated by inference alone.

(11) TuningActionExecutor: monitors the semantic reasoner (10), captures the inferred DB tuning actions and executes them in the DB according to the DBA (semi-automatic way) or the tool(automatic) choices.

(12) DB Tuning Actions: DB tuning actions received from TuningActionExecutor (11) through the interface (5) (by DBA) or the ontology (4) and semantic reasoner (10) (by agent), which must be applied to the database (1).

3.2 Outer-Tuning Ontology (OnDBTuning)

A subset of the concepts used in OnDBTuning is illustrated in Figure 2 with the capture of one of the queries of the TPC-H benchmark. Assume user *User_1* submits a DML statement *DML_1*. This is automatically classified as *DMLCommand* - *SingleStatement* - *QueryStatement* through rules defined in the ontology itself. This statement has as property the *hasDescription* SQL command derived from the TPC-H benchmark: *SELECT no_o_id FROM new_order WHERE no_w_id = 1 AND no_d_id = 1;*. The reasoner infers, using the rules, the presence of *SELECT*, *FROM* and *WHERE* clauses defined in such a statement. The complete domain ontology can be found in the URL: `https://www.ime.uerj.br/ondbtuning/`.

An example of a rule defined in the ontology can be seen in Figure 3. To use a given MV heuristic it is necessary to estimate the cost of creating the MV. This is defined, through the SWRL rule defined in the ontology, as being the cost of a simple scan of the MV plus the cost of recording the pages in secondary memory, estimated to be equivalent to 2 (two) times the number of hypothetical pages of a Hypothetical Materialized View (HMV) [de Oliveira 2015]. Thus, given the concept of HMV, if there is any HMV (line 1 - Figure 3), it produces a plan (line 2) of type RealExecutionPlan (line 3); such plan has the properties of hasExecutionCost (line 4) and hasHypotheticalPagesNumber (line 5) with values previously calculated respectively in the database's metadata and by another rule. Then, the cost is increased (line 7) by adding the product of the pages by two (line 6). If the total amount is greater than that zero (line 8), the property hasEstimatedCostCreationValue receives this value (line 9). We believe that the definition of the calculation by the rule using concepts from the DBA's own domain, improves the understanding of the rules used in the DB tuning decision of the tool as well as facilitates the definition of new rules and heuristics.

[1] `http://www.jessrules.com/`, last access: january, 2020.

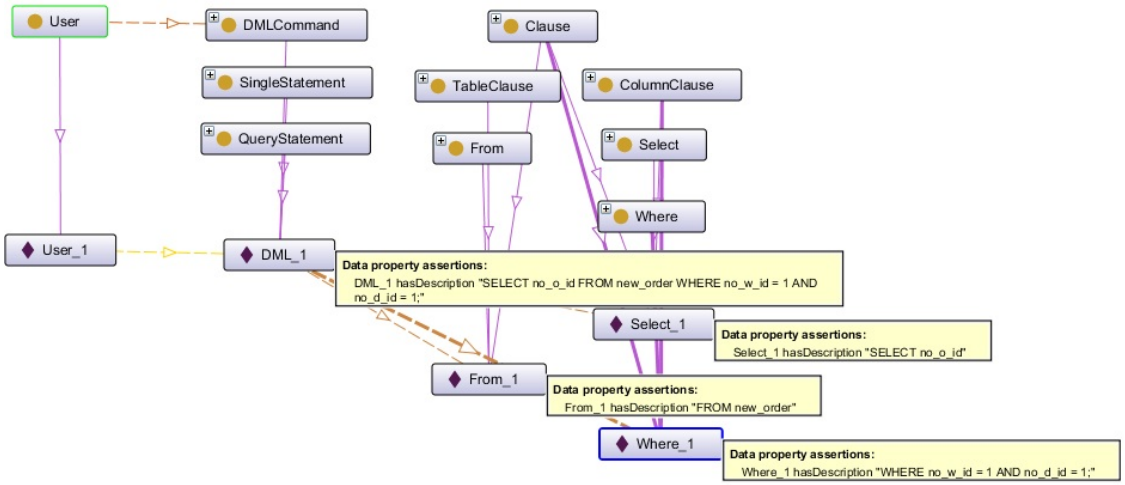

Fig. 2. Fragment of instantiated OnDBTuning

```
1  HypotheticalMaterializedView(?HMV) ^
2      produces(?HMV, ?Plan) ^
3  RealExecutionPlan(?Plan) ^
4      hasExecutionCost(?plan, ?QueryCost) ^
5      hasHypotheticalPagesNumber(?HMV, ?HipoPage) ^
6      swrlb:multiply(?HipoPageMult, ?HipoPage, 2) ^
7      swrlb:add(?Total, ?HipoPageMult, ?QueryCost) ^
8      swrlb:greaterThan(?Total, 0) →
9  hasEstimatedCostCreationValue (?HMV, ?Total)
```

Fig. 3. SWRL Rule that estimates the MV creation cost [de Oliveira et al. 2019]

3.3 Outer-Tuning in practice

Initially, the framework user can view the heuristics defined in the ontology and select those she wants to consider for future DB tuning suggestions. Subsequently, the user informs the tool the mode in which she wishes to work: semi-automatic or automatic (without human intervention). The tool starts capturing the workload in real-time and graphically displays when the DML command is executed in the DB and its duration in seconds. If the user wants more details on the command's execution, she may check further down the same screen (Figure 4).

If the user wants to follow the tuning actions analyzed and suggested by our tool, she may do so through the *Tuning actions* menu. Outer-Tuning shows a graph with the expected gain information with the tuning action (x-axis) and the estimated cost of creating the access structures (y-axis). The circle's size in the graph represents the number of SQL queries that the particular action can benefit from: the larger the circle's size, the greater the number of statements that benefit from the tuning action on the workload. The pop-up presented summarizes the proposed tuning action with the following information: expected gain, creation cost, type of action (*e.g.*, index or MV), and the number of statements benefited by the action.

It is important to remember that the purpose of our tool is not necessarily to have the best DB tuning heuristic, but to formally make available and add semantics to the existing heuristics. Experimental results demonstrate that Outer-Tuning is capable of simultaneously executing more than one heuristic for the same type of DB tuning strategy and providing information for the comparison of

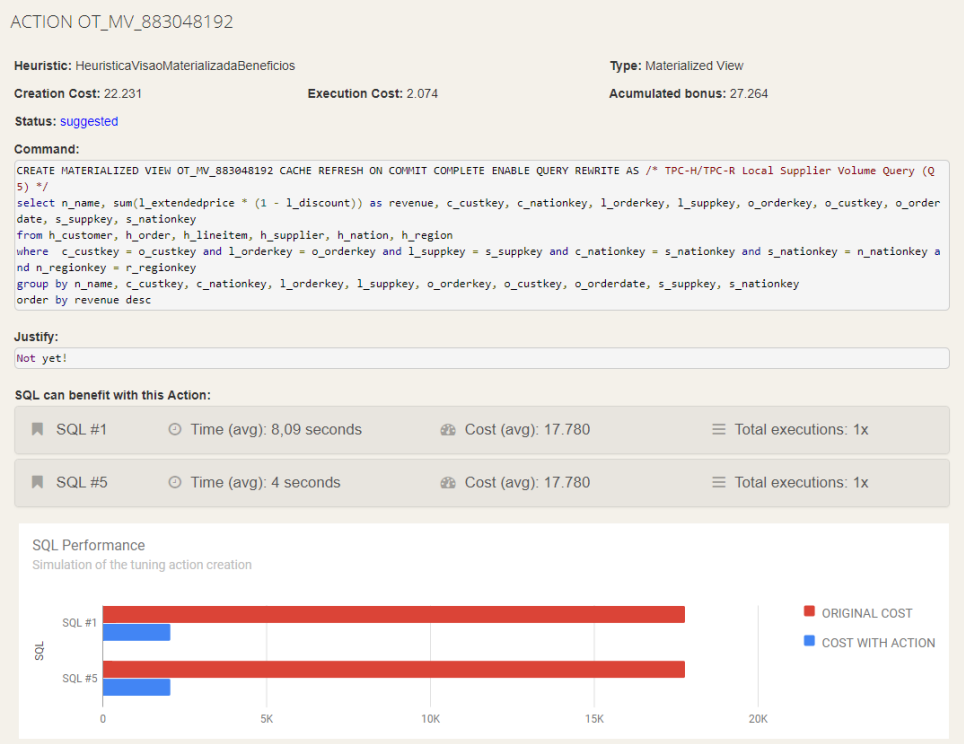

Fig. 4. Sample Screen of the Outer-Tuning Tool

inferred actions. The tool user selected three heuristics: (i) selection [Carvalho 2011] - responsible for suggesting HMV (not physically created yet) for each command submitted to DB; (ii) benefits [Morelli et al. 2012] - choose the selection heuristic MVs that can be physically created in the DB. This occurs when the accumulated benefit (execution plan cost gain) of the MV is greater than the cost of creating it; (iii) expectation [de Oliveira 2015] - similar to benefits, but without considering the frequency of command in the workload (non-accumulated benefit). In this case, it was assumed that if the cost of reading the result stored on disk is 50% less than the total cost of executing the original query, a MV could save time for the workload. The heuristics that propose MVs were executed, and their results were evaluated and compared using Outer-Tuning. For the generation of the workload during the tests, the benchmark TPC-H was used, conducive to evaluating MV selection tools, since it is OLAP. It should be noted that the tool presents both positive and negative evaluations. Some positive suggestions were implemented and brought benefits, as expected. More details are available at [de Oliveira et al. 2019].

4. OUTER-TUNING EXTENSIONS

Outer-Tuning was initially implemented with PostgreSQL and was later extended to include Oracle DBMS. With respect to this particular extension capability, we have decided to illustrate its extensibility showing the way we have included MySQL, a very popular open licence DB-Engine[2]. Besides, to facilitate our tool's use, both by users and by developers that might want to extend it, we show how to place it in a docker container.

[2]`https://db-engines.com/en/ranking\_trend/relational+dbms`, last access: march, 2021.

4.1 DBMS extension

The Outer-Tuning architecture, based on components, facilitates the inclusion of new Relational DBMSs. Initially, the JDBC driver needs to be downloaded and included in the framework's dependencies. Posteriorly, the abstraction class responsible for connecting to DBMS must be extended with connection details for the new RDBMS. Specific queries to the metadata of the new RDBMS need to be added to the *sql.properties* file. These queries get the *workload* (current queries submitted to the database), *names* and sizes of the referenced tables, referenced columns details, DDL syntax to create MV and execution plans for each workload query. These queries extract data that will be instantiated into the tuning ontology. The last step is to extend and implement all functions responsible to get data of execution plans (*e.g.,* getTotalCost, getTupleNumber, getDuration, and extractData).

Extension with MySQL DBMS. Outer-Tuning was extended with MySQL version 8.0.19. This version includes features that may help with other DB tuning strategies (e.g., invisible indexes) but the MV concept is still not available. Thus, in addition to the driver and the framework classes, other extensions were necessary that resulted in actions in the database system to allow Outer-Tuning to use heuristics that suggest MVs.

Following steps to extend Outer-Tuning with a new DBMS, we download the JDBC driver for MySQL[3]. Thus, the abstraction class was extended with MySQL connection details, describing the driver name and connection properties (database url, database name, username, and password).

Figure 5 shows specific queries to the metadata of MySQL that have been added to the sql.properties file. Line 1 shows the query to capture the workload; Lines 2 and 3 present the queries to get referenced table names and lengths, respectively; the query to get columns details is described in Line 4; Line 5 shows the code to create a simulated MV through a stored procedure (described below), and Line 6 has the explain query to capture the execution plan.

Regarding the last step, Figure 6 shows the main function to extract data from execution plans. These data include: total cost of the execution plan (this.cost), estimated number of rows returned (this.numRow), estimated size of a row (this.rowSize) among others. To simulate the concept of MV in MySQL and enable tuning heuristics, we will create tables derived from queries in other tables (base tables). For example, Figure 7 shows the statement responsible for creating a table derived from a query that retrieves the document number, name, and GPA of enrolled students. To update the tables that simulate the MVs, stored procedures will be created.

In order for the Outer-Tuning tool to create the database object that simulates the MV, we consider a stored procedure described in [Fedosseeva 2017]: it receives the view name and creates the derived table. A view is necessary to store the description (query) for data updates in the future. We also created the stored procedure responsible for updating the MV. According to the heuristic, this may be scheduled for execution.

4.2 Extension for Dockerization

The extension with Apache Maven aims to simplify the Outer-Tuning framework project's configuration. Maven is a project management tool that encompasses a set of standards, deals with the project's life cycle, has a dependency management system, and allows to run plugins in defined phases [Sonatype 2008]. Previously, there was no way to create the project executable (.war) without being done through an IDE, involving the entire test and build cycle. Adding Maven, this whole cycle and build are performed by Apache, allowing greater portability and speed when initiating any change in Outer-Tuning and Docker integration.

[3] `https://repo.maven.apache.org/maven2/mysql/mysql-connector-java/8.0.18/`, last access: march, 2021.

```
1. getSqlClauseToCaptureCurrentQueriesmysql=SELECT thread_id as 'pid', 'mysql' as
   'database_name', argument as 'sql', event_time as 'start_time' FROM mysql.general_log WHERE
   command_type ='Query'
2. getSqlTableNamesmysql=SELECT table_schema as 'schema', table_name as 'tablename',
   TABLE_ROWS as 'numberrrows', DATA_LENGTH as 'numberpages' FROM
   information_schema.tables where table_schema not in
   ('sys','performance_schema','information_schema') order by numberrrows desc;
3. getSqlTableLengthmysql=SELECT TABLE_ROWS as 'reltuples' FROM
   information_schema.tables where TABLE_NAME=?;
4. getSqlDetailsColumnsmysql=SELECT cl.ORDINAL_POSITION as ordernum,
   cl.COLUMN_NAME as columnname, coalesce(cl.IS_NULLABLE = false, true) as isnull,
   cl.DATA_TYPE as typefield, cl.COLUMN_DEFAULT as domainrestriction,
   coalesce(cl.COLUMN_KEY = 'PRI', true) as primarykey, coalesce(cl.COLUMN_KEY = 'UNI', true)
   as uniquekey, coalesce(kcl.COLUMN_NAME is not null, true) as foreignkey,
   kcl.ORDINAL_POSITION as foreignkey_fieldnum, kcl.COLUMN_NAME as foreignkey_name,
   kcl.TABLE_NAME as foreignkey_table, (select cl2.data_type from
   information_schema.COLUMNS cl2 where cl2.TABLE_NAME = kcl.TABLE_NAME and
   cl2.TABLE_SCHEMA = kcl.TABLE_SCHEMA and cl2.COLUMN_NAME =
   kcl.COLUMN_NAME) as foreignkey_type from information_schema.COLUMNS cl LEFT JOIN
   information_schema.KEY_COLUMN_USAGE kcl on (kcl.REFERENCED_TABLE_NAME =
   cl.TABLE_NAME and kcl.REFERENCED_TABLE_SCHEMA = cl.TABLE_SCHEMA and
   kcl.REFERENCED_COLUMN_NAME = cl.COLUMN_NAME)where cl.TABLE_SCHEMA =
   '$schema$' and cl.TABLE_NAME = '$table$' ORDER BY ordernum;
5. getDDLCreateMVmysql=call create_matview('$nameMV$');
6. getPlanQueryMySQL=EXPLAIN FORMAT=JSON $QUERY$
```

Fig. 5. Part of the changes - Outer-Tuning with MySQL

```
1. public PlanMySQL(String plan, Date time) { ...
2.    try { ...
3.          this.cost =
   Optional.ofNullable(explain.getQueryBlock()).map(Explain.QueryBlock::getCostInfo).m
   ap(Explain.CostInfo::getQueryCost).map(Double::parseDouble).orElse(0D).longValue();
4. ...
5.          this.numRow =
   nestedLoop.stream().map(Explain.Table::getTableInfo).mapToInt(Explain.TableInfo::getR
   owsExaminedPerScan).sum();
6.          this.rowSize =
   nestedLoop.stream().map(Explain.Table::getTableInfo).mapToInt(Explain.TableInfo::getR
   owsProducedPerJoin).sum();
7.  ...   } catch (JsonSyntaxException e) { … }
8. }
```

Fig. 6. Main function to get data from the execution plan

In addition to Maven, we consider the concept of a container within the Docker[4] tool. We may then separate the applications from the execution infrastructure, making the process of testing and application deployment faster. Docker is a platform that allows us to control, at the software level, all containers that are created and executed. With this dockerization, the Outer-Tuning environment will have three different containers: MySQL, Outer-Tuning framework and a script to run the workload test. To orchestrate these containers, a docker-compose.yml file was created with the settings.

[4]`https://docs.docker.com/get-started/overview/\#the-docker-platform`, last access: march, 2021.

```
1. CREATE TABLE mv_student_gpa AS
2. SELECT s.documentNumber, s.name, sum(cs.credit*cs.grade)/sum(cs.credit) AS gpa
3. FROM student s, class_student cs, class c
4. WHERE s.id=cs.id_student and cs.id_class=c.id and cs.end_date is not null
5. GROUP BY s.documentNumber, s.name;
```

Fig. 7. Example of creating a simulated materialized view (Derived table)

Due to the dockerization, the installation and use of Outer-Tuning are now straightforward compared with previous versions. Once Maven, Docker, and the docker-compose are present, now the user can clone (or download) the Outer-Tuning project to build and run their containers with few and simple steps. More details about the extension and the download are available at `https://github.com/raphaelfmarins/outer\_tuning`.

5. CONCLUSIONS

We have presented the Outer-Tuning tool, a framework to support decision making while tuning relational DB systems. It is possible to visualize the use of Outer-Tuning in a semi-automatic or automatic way, enabling tuning experts to interact with it. Our tool's innovative contribution is to offer transparency and reliability about the alternatives available for possible scenarios in the database system through concrete justifications for the decisions that are ontology-based and semantically defined. We also have shown some extensibility features, including DBMS instantiation and dockerization.

As current and future works, Outer-Tuning is being extended to SQL Server DBMS and will further include the justifications for the DBAs' decisions when the semi-automatic mode is on. We also intend to develop the ontology used by the tool to contemplate the concepts defined in Tun-OCM [Almeida et al. 2021], where management of database system configurations can be more controlled and recorded.

REFERENCES

Alhadi, N. and Ahmad, K. Query tuning in oracle database. *Journal of Computer Science* 8 (11): 1889, 2012.

Almeida, A. C., Baião, F., Lifschitz, S., Schwabe, D., and Campos, M. L. M. Tun-ocm: A model-driven approach to support database tuning decision making. *Decision Support Systems* vol. 145, pp. 113538, 2021.

Almeida, A. C., Campos, M. L. M., Baião, F., Lifschitz, S., de Oliveira, R. P., and Schwabe, D. An ontological perspective for database tuning heuristics. In *International Conference on Conceptual Modeling*. Springer, Springer, pp. 240–254, 2019.

Almeida, A. C., Haeusler, E. H., Lifschitz, S., de Oliveira, R. P., and Schwabe, D. Outer-tuning: Automatic self-tuning based on ontology (in portuguese). In *Demos Session, Proceedings of the Brazilian Symposium on Databases (SBBD)*. SBC, pp. 29–34, 2018.

Bruno, N. *Automated Physical Database Design and Tuning*. CRC Press, 2011.

Carvalho, A. W. *Automatic Creation of Materialized Views in Relational DBMSs (In Portuguese)*. M.S. thesis, PUC-Rio, Brazil, 2011.

de Oliveira, J., Loja, L. F., da Costa, S. L., and Neto, V. G. An information systems component for business process management (in portuguese). In *Proceedings of the VII Brazilian Symposium on Information Systems (SBSI)*. SBC, Porto Alegre, RS, Brasil, pp. 250–261, 2011.

de Oliveira, R. P. *Ontology-Based Tuning: The Case of Materialized Views (In Portuguese)*. M.S. thesis, PUC-Rio, Brazil, 2015.

de Oliveira, R. P., Baião, F., Almeida, A. C., Schwabe, D., and Lifschitz, S. Outer-tuning: an integration of rules, ontology and rdbms. In *Proceedings of the XV Brazilian Symposium on Information Systems*. ACM, pp. 1–8, 2019.

Dias, K., Ramacher, M., Shaft, U., Venkataramani, V., and Wood, G. Automatic performance diagnosis and tuning in oracle. In *CIDR*. www.cidrdb.org, pp. 84–94, 2005.

Fedosseeva, A. Speeding Up Mysql Using Materialized Views. https://medium.com/@anna.f/speeding-up-mysql-by-using-materialized-views-282ecbd3a53f, 2017.

Gamma, E., Johnson, R., Helm, R., Johnson, R. E., Vlissides, J., et al. *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH, 1995.

Goasdoué, F., Karanasos, K., Leblay, J., and Manolescu, I. View selection in semantic web databases. *Proc. VLDB Endow.* 5 (2): 97–108, Oct., 2012.

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., Dean, M., et al. Swrl: A semantic web rule language combining owl and ruleml. *W3C Member submission* 21 (79): 1–31, 2004.

Khosla, R. and Ichalkaranje, N. *Design of intelligent multi-agent systems: human-centredness, architectures, learning and adaptation*. Springer-Verlag Berlin Heidelberg, 2005.

Morelli, E., Almeida, A., Lifschitz, S., Monteiro, J. M., and Machado, J. Autonomous re-indexing. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. ACM, pp. 893–897, 2012.

Shasha, D. and Bonnet, P. *Database tuning: principles, experiments, and troubleshooting techniques*. Elsevier, 2002.

Sonatype. *Maven: The Definitive Guide*. " O'Reilly Media, Inc.", 2008.

Zhang, B., Van Aken, D., Wang, J., Dai, T., Jiang, S., Lao, J., Sheng, S., Pavlo, A., and Gordon, G. J. A demonstration of the ottertune automatic database management system tuning service. *Proceedings of the VLDB Endowment* 11 (12): 1910–1913, 2018.

Zhang, J., Liu, Y., Zhou, K., Li, G., Xiao, Z., Cheng, B., Xing, J., Wang, Y., Cheng, T., Liu, L., 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*. ACM, pp. 415–432, 2019.