Integration of the OnDBTuning Ontology into the OuterTuning Framework

Eric Ruas Leão[1], Edward Hermann Haeusler[1] and Sergio Lifschitz[1,†]

[1]Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), Rio de Janeiro/RJ, Brazil

Abstract

This paper presents the integration of the *OnDBTuning* ontology into the *OuterTuning* framework to deliver an explainable, semi-automatic approach to relational database tuning. OnDBTuning encodes expert knowledge about indexes, materialised views, and other optimisation actions as SPARQL rules, while OuterTuning captures workloads, runs what-if simulations, and provides an interactive GUI for DBAs. By linking both tools through a RESTful API that converts workload metadata to RDF and returns inferred actions, we replace OuterTuning's rigid heuristics with ontology-driven reasoning. Proprietary SPIN functions were reimplemented in standard SPARQL, removing license restrictions and increasing portability. Experiments with the TPC-H benchmark (scale factor 0.01) show that a curated subset of ontology-suggested indexes improves the Queries-per-Hour metric by 295% over the no-index baseline, while using only 25% of the storage consumed by a reinforcement-learning approach. The resulting architecture offers transparency, modularity, and tangible performance gains, making it suitable for production environments and database-tuning education.

Keywords

Ontology, OnDBTuning, OuterTuning, Database Tuning, SPARQL, Semantic Web

1. Introduction

Database tuning is crucial for optimizing the performance of relational database systems, directly influencing query response times and transaction efficiency [1]. Traditionally, database administrators (DBAs) rely on their expertise and automatic tools to analyze the system's behavior and propose adjustments such as index creation, memory parameter configurations, and query optimizations. However, these methods often involve subjective decisions, lack transparency, and can be inconsistent or limited in adaptability.

To overcome these limitations, ontologies have emerged as a promising approach for fine-tuning databases by explicitly representing domain knowledge and enabling automatic inference through predefined rules [2][3]. The OnDBTuning ontology, designed explicitly for relational database tuning, formalizes and structures this knowledge, enabling adaptive suggestions for various database scenarios[4]. Concurrently, OuterTuning provides DBAs with a modular framework capable of real-time workload monitoring and transparent decision support through visual recommendations [5].

However, OuterTuning initially employed hard-coded heuristics, limiting flexibility and adaptability. Integrating OnDBTuning into OuterTuning addresses this issue by providing dynamic and adjustable inferences, enhancing both tools' capabilities. This integration required reimplementing proprietary SPIN functions in standard SPARQL, which removed dependencies on licensed components and increased portability. The result is a comprehensive, automated solution capable of transparent and justifiable tuning actions, suitable for both professional and educational contexts. In this sense, this work aims to integrate these two previous studies to create a robust tool combining the strengths of OnDBTuning and OuterTuning.

Proceedings of the 18th Seminar on Ontology Research in Brazil (ONTOBRAS 2025) and 9th Doctoral and Masters Consortium on Ontologies (WTDO 2025), São José dos Campos (SP), Brazil, September 29 – October 02, 2025.

*Corresponding author.

†These authors contributed equally.

eric.leao@inf.puc-rio.br (E. R. Leão); hermann@inf.puc-rio.br (E. H. Haeusler); sergio@inf.puc-rio.br (S. Lifschitz)

0000-0002-4999-7476 (E. H. Haeusler); 0000-0003-3073-3734 (S. Lifschitz)

Structure. Section 2 introduces the foundations and design choices; Section 3 reviews related work; Section 4 describes the architecture and experimental setup; Section 5 presents results and analysis; Section 6 concludes and outlines future work.

2. Foundations

Database tuning is a critical task to enhance the performance of relational database systems. It involves selecting and applying actions—such as index creation, parameter adjustment, and query rewriting—to reduce execution time and resource consumption. Traditional tuning strategies often rely on manual decisions based on the DBA's experience or automated tools that behave as black boxes. These approaches lack transparency, reproducibility, and adaptability, especially in dynamic and heterogeneous workloads.

To address these challenges, the OnDBTuning provides a formal and extensible model of the tuning domain, including concepts such as tables, columns, queries, indexes (simple, composite, partial), materialized views, and cost-based decision rules. OnDBTuning adopts Semantic Web standards such as RDF and OWL for knowledge representation. SPARQL is employed not only to query RDF data but also to encode inference rules that capture tuning heuristics, such as the automatic creation of indexes and materialized views [2].

Initially, the inference mechanism of OnDBTuning relied on TopSPIN, a proprietary SPIN engine integrated into TopBraid Composer [6]. However, these dependencies were removed due to licensing constraints and the need for broader applicability. Proprietary functions like `spif:split`, `spif:countMatches`, `spif:trim`, and `spif:localName` were rewritten using standard SPARQL 1.1 constructs or replaced with equivalent custom logic. This shift enabled the use of open technologies such as Apache Jena [7] and the SPIN RDF vocabulary directly, without proprietary extensions.

Complementing the ontology, the OuterTuning framework is a Java-based application that offers a flexible environment for simulating, monitoring, and evaluating the impact of tuning actions. It consists of components such as a workload analyzer, rule execution engine, graphical interface, and simulation orchestrator. OuterTuning provides visual feedback on tuning suggestions, enabling users to compare different scenarios using real cost estimates interactively.

Originally, OuterTuning employed hardcoded SQL heuristics, limiting its extensibility. Integrating with OnDBTuning, the system was restructured to support semantic reasoning and knowledge-driven decisions. A RESTful Web API was implemented to automate the metadata and SQL workloads transformation into RDF, allowing OuterTuning to act as a consumer of ontology-based recommendations. This architectural improvement promotes modularity, maintainability, and compatibility with other Semantic Web applications.

OnDBTuning and OuterTuning offer a robust, explainable, and portable solution for database tuning. Their integration provides an operational benefit in terms of performance optimization and is a didactic tool for teaching database administration and semantic web reasoning.

3. Related Work

The state of the art in relational database tuning includes both commercial tools and academic approaches. Commercial solutions such as the PostgreSQL Workload Analyzer (PoWA) offer monitoring and tuning support based on system statistics and query performance [8]. While useful for performance inspection and index recommendation, these tools are often limited to specific environments, lack formal semantic reasoning, and provide limited transparency, making them inadequate for pedagogical use or research extensibility.

Academic efforts increasingly explore advanced optimization techniques, including Bayesian optimization, reinforcement learning, and, more recently, large language models (LLMs). These approaches aim to dynamically automate and adapt tuning actions, often by learning from historical workloads.

Bayesian Optimization. Tools such as OtterTune and ResTune apply Bayesian optimization to automatically adjust configuration parameters in relational databases [9][10]. OtterTune, for example, uses Gaussian process models to recommend configurations based on historical workload data, while ResTune extends this idea with constrained optimization techniques to address latency and throughput requirements.

Reinforcement Learning. SmartIX and rCOREIL represent efforts to apply reinforcement learning (RL) to the automatic tuning of indexes [11][12]. In these systems, an RL agent explores different index configurations and learns which ones yield better performance over time. These approaches have shown strong potential in dynamic and heterogeneous workloads but remain complex to interpret and challenging to integrate with human-readable reasoning mechanisms.

Semantic Approaches. While LLMs and ML-based systems aim for adaptivity, they often lack transparency. Semantic Web approaches, including OnDBTuning, fill this gap by encoding expert knowledge in a formal ontology, enabling rule-based inference through SPARQL. Although fewer in number, such systems offer interpretable and customizable alternatives to opaque ML pipelines.

In contrast to previous ontology-based efforts that focused on schema mapping or query reformulation, OnDBTuning is specifically designed to capture fine-tuning strategies. Combined with OuterTuning—a Java-based tool for simulating and visualizing tuning actions—the integrated solution enables an open, extensible, and explainable workflow that bridges theoretical reasoning and practical experimentation.

4. The Experiments

We implemented a complete experimental setup combining semantic inference with real-time workload analysis to validate the proposed integration between the OnDBTuning ontology and the OuterTuning framework[1]. This integration was achieved by extending the architecture of OuterTuning with a RESTful Web API, which is responsible for mediating communication between the framework and the inference engine. The API receives a JSON payload containing metadata such as table structures and executed SQL queries, and returns a list of tuning recommendations including rule identifiers, associated queries, expected benefit (bonus), and the corresponding SQL command.

The integration followed the Strangler Pattern [13], initially allowing both legacy and new components to run in parallel. This ensured a safe migration by comparing inference outputs and validating their consistency before deactivating the static logic previously embedded in the system. The API architecture allows for modularity and extensibility, allowing it to incorporate other ontologies or inference mechanisms in the future.

Figure 1 presents the expanded architecture of the OuterTuning framework after integration with OnDBTuning.

The architecture is composed of the following components:

Components. As shown in Fig. 1, (1) *Database* — JDBC access for workload monitoring and action execution; (2) *OuterTuning Framework* — orchestration layer; (3) *Function Libraries* — extract workload features; (4) *OnDBTuning Ontology* — domain concepts and SPARQL rules; (5) *Web API* — JSON↔RDF mediation and suggestion delivery; (6) *Interface* — DBA inspection/validation; (7) *JDBC Connections* — DB interaction; (8) *WorkloadCollector* — queries, plans and frequencies; (9) *FunctionExecutor* — applies libraries to features; (10) *ConceptInstantiator* — creates RDF instances; (11) *SemanticReasoner* — rule execution over the ontology; (12) *TuningActionExecutor* — applies actions (auto or user-guided); (13) *Execution Feedback* — measures outcomes for iterative refinement.

We use the TPC-H benchmark to evaluate the framework's effectiveness, a well-established decision support workload [14]. We generated 660 queries by creating 30 variants of each of the 22 standard TPC-H queries, covering a wide range of complexity, including multiple joins, filters, and aggregations. These queries were used to simulate realistic database usage scenarios and to test the framework's capability to extract metadata, instantiate concepts, and infer tuning actions.

[1]Source code available at https://github.com/EricRLeao1311/outer_tuning/tree/outertuning_expandido

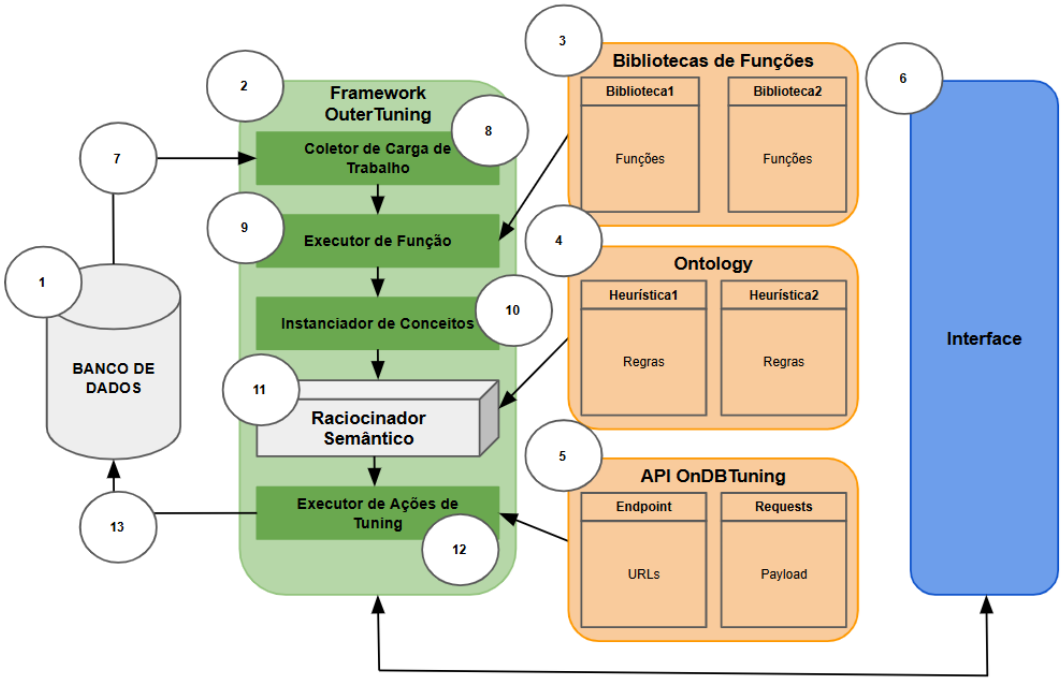

Figure 1: Expanded architecture of the OuterTuning framework

Four rule types were evaluated: *RuleHypSimpleIndex* (for suggesting single-column indexes) , *RuleHypCompositeIndex* (for generating composite indexes that combine multiple columns), *RuleSimplePartialIndex* (for creating partial indexes with specific `WHERE` clause filters), and *RuleHypViewAdapted* (for generating materialized views based on common aggregations). The evaluation included metrics such as the number of times each rule was triggered, the average bonus per action, and the number of queries each suggested tuning action benefited. Results showed that index-based rules—straightforward and composite indexes—provided broader impact, while materialized views and partial indexes were more specific and restrictive in their applicability.

Overall, the experimental integration proved the feasibility and effectiveness of combining declarative semantic reasoning with a workload-aware tuning framework. The modular API design and inference pipeline allow future extension and promote the adoption of explainable optimization mechanisms in both academic and professional database tuning scenarios.

5. Results

To assess the effectiveness of the integrated framework, we conducted a comparative performance evaluation using the TPC-H benchmark with a scale factor of 0.01 (approximately 10MB). The evaluation considered both the quality of tuning suggestions and their impact on system performance under different index configurations.

5.1. Evaluation Metrics

We adopted standard TPC-H performance metrics to quantify the benefits of each tuning strategy:

- **Power:** Measures performance under a single-stream workload (sequential query execution). It is computed as:

$$\text{Power} = \frac{3600 \times \text{SF}}{\left(\prod_{i=1}^{n} t_i\right)^{1/n}}$$

  where t_i represents the execution time of each operation (queries and refresh functions), and SF is the scale factor.

- **Throughput:** Measures performance under multi-stream execution (parallel queries). It is calculated as:
$$\text{Throughput} = \frac{S \times N \times 3600 \times \text{SF}}{T}$$
where S is the number of streams, N the number of operations per stream, and T the total elapsed time.
- **QphH (Queries per hour):** The geometric mean of Power and Throughput:
$$\text{QphH} = \sqrt{\text{Power} \times \text{Throughput}}$$
- **Index Size:** Total disk space consumed by all indexes, measured in megabytes.

5.2. Compared Configurations

We compared the following tuning strategies:

- **No Index:** Baseline with no indexes applied.
- **DBA Expert:** Manually selected indexes simulating expert knowledge.
- **POWA:** Automatic recommendations using PostgreSQL Workload Analyzer.
- **rCOREIL:** Reinforcement learning-based tuning with dynamic index combinations.
- **OuterTuning Simple / Complete:** configurations applying either only simple indexes (Simple) or the full set of simple plus composite indexes (Complete) as our framework recommends.
- **OuterTuning Simple Selected / Complete Selected:** Subsets of indexes manually validated using the GUI to balance performance and index size.

5.3. Results and Analysis

Table 1 presents the evaluation results for each configuration.

Table 1
performance of tuning strategies with TPC-H (SF=0.01)

Configuration	Power	Throughput	QphH	Index Size (MB)
OuterTuning Complete Selected	**2013.96**	**1398.81**	**1678.44**	**4.93**
OuterTuning Simple Selected	1693.14	724.48	1107.54	2.71
rCOREIL	1995.72	547.45	1045.25	20.22
OuterTuning Simple	1432.78	443.78	797.40	7.00
OuterTuning Complete	1811.23	257.70	683.19	53.31
DBA Expert	1374.61	172.34	486.73	1.58
POWA	1219.53	159.37	440.86	2.07
No Index	1007.00	178.96	424.51	0.00

The *OuterTuning Complete Selected* configuration outperformed all others, achieving a QphH of 1678.44 — a 295% improvement over the *No Index* baseline. It also maintained a compact index footprint of only 4.93 MB, illustrating that selective application of high-impact indexes via the graphical interface leads to an optimal trade-off between performance and storage cost.

Although *rCOREIL* achieved a comparable Power score, its QphH was 38% lower than that of *OuterTuning Complete Selected*, while using over four times the index size. This underscores the effectiveness of explainable, rule-based inference combined with human-in-the-loop filtering.

The *OuterTuning Complete* configuration, which indiscriminately applied all suggested indexes, showed worse results than its selected variant. This reinforces the importance of prioritizing quality over quantity in index selection. Similarly, traditional strategies like *POWA* and *DBA Expert* delivered lower QphH scores due to their reliance on simple indexes and lack of adaptive reasoning.

In summary, the results validate the advantages of the proposed framework in achieving high performance while maintaining interpretability, modularity, and efficient resource usage.

6. Conclusion

This work presented the integration of the OnDBTuning ontology into the OuterTuning framework, resulting in a semantic, extensible, and explainable system for semi-automatic tuning of relational databases. By combining rule-based reasoning with real-time workload monitoring, the integrated framework can generate interpretable tuning recommendations — such as indexes and materialized views — and apply them in a modular, user-guided manner.

The proposed Web API and its decoupled architecture enabled seamless integration with OuterTuning, allowing dynamic inference using SPARQL and eliminating dependencies on proprietary tools. The results obtained through the TPC-H benchmark demonstrated that the *OuterTuning Complete Selected* configuration achieved superior performance in both Power and Throughput, while maintaining low storage overhead. Compared to traditional tools such as POWA and reinforcement learning approaches like rCOREIL, our approach provided competitive — and in many cases superior — results, with greater transparency and flexibility.

However, despite the promising findings, future work must address some limitations to ensure robustness and generalizability. First, it is necessary to compare the system with a broader set of tuning tools that support simple indexes, composite indexes, and materialized views. This will allow for a more complete and balanced evaluation of the framework's inference capabilities.

Second, the current experiments were conducted using a single database instance with a scale factor of 0.01 (approximately 10MB), and the rCOREIL results were taken from its published paper. A more rigorous evaluation should involve running both tools under the same conditions and on diverse databases with varying workloads and sizes. This would help ensure fair and unbiased comparisons and confirm the scalability and adaptability of the proposed solution.

As future work, we plan to address these gaps by extending the benchmarking suite, increasing data scale, and expanding the range of inference strategies integrated into the framework to strengthen its reliability and practical applicability in real-world database environments.

Acknowledgments

This study was partially funded by the CAPES Institutional Grant for PUC-Rio. Sergio Lifschitz is partially funded by CNPq Research Grant 313504/2025-3, Edward H. Haeusler is partially funded by CNPq Research Grant 309287/2023-5, and FAPERJ APQ1 Grant E-26/210.258/2019.249292. The author is supported by a CAPES scholarship and gratefully acknowledges this support.

Declaration on Generative AI

While preparing this work, the authors used GPT-4o to: Grammar, spelling check, and Text translation. After using this tool, the authors reviewed and edited the content as needed and assume full responsibility for the publication's content.

References

[1] D. Shasha, P. Bonnet, Database Tuning: Principles, Experiments and Troubleshooting Techniques, Morgan Kaufmann, San Francisco, 2002.

[2] A. C. Almeida, M. L. M. Campos, F. Baião, S. Lifschitz, R. P. de Oliveira, D. Schwabe, An ontological perspective for database tuning heuristics, in: A. H. F. Laender, B. Pernici, E. Lim, J. P. M. de Oliveira (Eds.), Conceptual Modeling - 38th International Conference, ER 2019, Salvador, Brazil, Proceedings, volume 11788 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 240–254.

[3] S. Staab, R. Studer, Handbook on Ontologies, Springer, 2010.

[4] L. de Sá Silva Perciliano, V. dos Santos, F. Baião, E. H. Haeusler, S. Lifschitz, A. C. Almeida, Inferencing Relational Database Tuning Actions with OnDBTuning Ontology, in: Procs Brazilian Symposium on Databases, SBBD, Rio de Janeiro, Brazil (Online), 2021, pp. 157–168.

[5] R. P. de Oliveira, F. Baião, A. C. Almeida, D. Schwabe, S. Lifschitz, Outer-tuning: an integration of rules, ontology and RDBMS, in: F. G. Rocha, I. Vasconcelos, R. P. dos Santos, D. Viana, S. de Avila e Silva (Eds.), Proceedings of the XV Brazilian Symposium on Information Systems, SBSI, Aracaju, Brazil, 2019, pp. 60:1–60:8.

[6] SPIN RDF, Spin – sparql inferencing notation, 2024. https://spinrdf.org/.

[7] Apache Jena, Apache jena documentation, 2024. https://jena.apache.org/.

[8] Dalibo, PoWA: PostgreSQL Workload Analyzer — documentation, https://powa.readthedocs.io/, 2025. Version 5.0.0, accessed 2025-07-13.

[9] D. V. Aken, A. Pavlo, G. J. Gordon, B. Zhang, Automatic database management system tuning through large-scale machine learning, in: S. Salihoglu, W. Zhou, R. Chirkova, J. Yang, D. Suciu (Eds.), Proceedings of the 2017 ACM International Conference on Management of Data SIGMOD, Chicago, IL, USA, 2017, pp. 1009–1024.

[10] X. Zhang, H. Wu, Z. Chang, S. Jin, J. Tan, F. Li, T. Zhang, B. Cui, Restune: Resource oriented tuning boosted by meta-learning for cloud databases, in: G. Li, Z. Li, S. Idreos, D. Srivastava (Eds.), SIGMOD '21: International Conference on Management of Data, Virtual Event, China, 2021, pp. 2102–2114.

[11] G. P. Licks, J. M. C. Couto, P. de Fátima Miehe, R. D. Paris, D. D. A. Ruiz, F. Meneguzzi, Smartix: A database indexing agent based on reinforcement learning, Applied Intelligence 50 (2020) 2575–2588.

[12] D. Basu, Q. Lin, W. Chen, H. T. Vo, Z. Yuan, P. Senellart, S. Bressan, Regularized cost-model oblivious database tuning with reinforcement learning, Transactions on Large Scale Data Knowledge Centered Systems 28 (2016) 96–132.

[13] S. Newman, Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith, O'Reilly Media, 2020.

[14] Transaction Processing Performance Council, TPC-H benchmark specification, revision 2.17.3, 2023. https://www.tpc.org/tpch/.