# Can LLMs Replace the DBA? Evaluating Ontology-Driven Prompting for Relational Database Tuning Tasks

**Eric Ruas Leão**[1], **Sergio Lifschitz**[1], **Ana Carolina Almeida**[2], **Edward Hermann Haeusler**[1]

[1] Departamento de Informática – Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)
Rio de Janeiro – RJ – Brazil

[2]Departamento de Computação – Universidade do Estado do Rio de Janeiro (UERJ)
Rio de Janeiro – RJ – Brazil

{eleao,sergio,hermann}@inf.puc-rio.br, ana.almeida@ime.uerj.br

***Abstract.*** *Relational database tuning traditionally relies on specialized heuristics and expert intuition, making the process difficult to scale, audit, and reproduce. While recent research has explored autonomic and AI-based systems to assist Database Administrators, the emergence of Large Language Models introduces a new paradigm for automated tuning. This paper investigates the extent to which LLMs can augment or replace traditional DBA expertise in complex, knowledge-dependent tuning tasks. We evaluate tuning recommendations generated by two distinct LLMs across three model variants each, comparing zero-shot performance against ontology-driven prompting. Using a PostgreSQL environment with a TPC-H-like workload, we measure the power, throughput, and storage footprint of the recommended structures. Our results indicate that expert knowledge—whether from an experienced DBA or a formal ontology—is not merely a source of domain vocabulary. Ultimately, the ontology acts as a critical precision mechanism that codifies the 'rules of thumb' inherent to expert tuning, effectively grounding the LLM's generative capacity and optimizing the trade-off between performance and storage footprint.*

## 1. Introduction

Relational database tuning remains a cornerstone of performance for both analytical and transactional applications. Despite decades of advancement in query optimizers and self-managing features, the selection of optimal physical structures—such as indexes and materialized views—still relies heavily on domain-expert knowledge. This expertise comprises nuanced heuristics and *rules of thumb* that are highly sensitive to the specific database engine, version, and shifting workload patterns.

The recent emergence of Large Language Models (LLMs) has sparked a new frontier in database research, centering on the potential for these models to augment or even replace traditional Database Administrator (DBA) expertise. While LLMs show remarkable generative capabilities, it remains unclear whether they can replicate the specialized logic and cost-benefit intuition of established experts, whether those experts are seasoned human DBAs or automated, heuristic-based support systems. To bridge the gap between an LLM's general linguistic patterns and specialized domain knowledge, formal structures such as ontologies can serve as a grounding mechanism.

This paper investigates the extent to which LLMs can replicate or surpass traditional expert intervention in complex tuning tasks. We conduct an extensive experimental study across 96 distinct scenarios, evaluating three model variants against established expert-knowledge benchmarks. By comparing standard LLM prompting against ontology-driven inputs across this broad experimental space, we measure the models' ability to approximate expert-level decisions. The evaluation is conducted using a representative real-world Database Management System (DBMS) and a TPC-H-like workload, measuring results through multiple lenses: throughput, storage footprint, and efficiency.

The primary contributions of this work are threefold. First, we provide a large-scale experimental evaluation of LLM-generated tuning recommendations under different prompt and ontology conditions. Second, we present a comparative analysis of model scale, investigating how different model capacities use structured domain knowledge. Finally, we conduct an efficiency analysis showing how ontology-guided prompting affects the trade-off between performance gains and storage footprint.

The remainder of this article is organized as follows: after this brief introduction, Section 2 presents related research work, and Section 3 formulates our main hypothesis, which will be discussed throughout this manuscript. Then, Section 4 describes the experimental analysis and metrics, as well as some research questions. Finally, Section 5 presents our practical experiments, Section 6 discusses the results obtained, and Section 7 presents conclusions and possible future work.

## 2. Related Work

The literature on automated physical database design shows that choosing partitions, indexes, knobs, and other tuning strategies is far from trivial [Liu et al. 2026, Wu et al. 2024, Zhao et al. 2023]. The complexity of the problem arises from interactions among queries, predicate selectivity, update costs, overlap among candidate structures, interactions among multiple knobs, and internal optimizer behavior.

Previous work on database tuning, either autonomous or not, has highlighted two central points. First, database tuning should be driven by the overall workload rather than by isolated heuristics applied to individual queries. Second, the quality of a recommendation depends on the balance between read performance gains and associated costs, such as storage and maintenance. In other words, a larger set of structures is not necessarily better than a smaller and more carefully selected one.

The authors in [Mozaffari et al. 2024] present a systematic literature review on automatic database schema design and tuning, which the interested reader can refer to for a detailed study of these issues. There are many researchers in the Brazilian community who have published interesting works concerning database tuning issues, e.g. [de Araújo et al. 2014, Alencar et al. 2019, Fuentes et al. 2018].

Some authors have explored conceptual models in the relational database domain [Metamodel 2003, Calero et al. 2006, Ouared et al. 2016, de Aguiar et al. 2018]. These works describe objects in relational databases (e.g., tables, columns, data types, data structures), providing greater knowledge and transparency into the relationships among these objects.

The second group of studies addresses the use of conceptual models for database

tuning [Almeida et al. 2019, Almeida et al. 2021, Perciliano et al. 2021]. This group complements the previous one by formalizing database tuning concepts through an ontological representation and inference rules for the knowledge representation of tuning actions.

This research line has evolved toward broader problems, such as the combination and selection of tuning actions [Oliveira et al. 2022] and cost-benefit-guided tuning decisions [Souza and Lifschitz 2022]. These works reinforce an important point for this paper: tuning should not be viewed merely as the generation of candidate structures, but as a reasoning process that must account for utility, interaction, and constraints.

LLMs have recently been investigated for several database tasks. For example, an LLM-Driven Index Tuning approach [Wang et al. 2026]. DB-GPT [Xue et al. 2024] presents a systems-oriented view in which LLMs can be used with prompting, fine-tuning, and specialized mechanisms for query rewriting and index tuning. In parallel, there are several studies that use LLMs to recommend database tuning knobs [Dou et al. 2026, Giannakouris and Trummer 2025, Lao et al. 2025, Li et al. 2025, Li et al. 2024]. Some works use knowledge of DBMS manuals and forum discussions to enrich tuning decisions with LLM [Lao et al. 2025].

However, an important gap remains: there are still few studies that isolate the effect of structured domain knowledge, in this case, an ontology or knowledge from an experienced DBA, on physical tuning recommendations, from those suggestions generated by LLMs. Our work is positioned exactly at this intersection. Rather than studying either LLMs or ontology-driven modeling, we evaluate the effect of combining the two to explore whether human- or systems-based DBA can already be replaced by AI mechanisms, or whether there is still a long way to go in this direction.

## 3. Proposed Approach

The central hypothesis of this work is that a tuning ontology can function as a grounding mechanism for LLMs, reducing redundant, weakly selective, or poorly workload-aligned recommendations. More specifically, we expect the ontology to help the model:

1. prioritize recurring join and filtering patterns;
2. avoid suggestions that are redundant with already existing indexes;
3. distinguish when simple, composite, and partial indexes are more appropriate; and
4. reduces the tendency to suggest structures for small tables or low-selectivity predicates.

This hypothesis does not imply that the ontology will always improve every model. On the contrary, its effect may depend on the LLM's ability to use the knowledge it receives correctly. For this reason, the experiment was designed as an ablation study.

### 3.1. Experimental pipeline

Figure 1 summarizes the pipeline adopted in this work. The process receives schema metadata, workload information, task instructions, and, optionally, the OnDBTuning ontology. The LLM generates candidate structures. These structures are then converted

into concrete tuning definitions and evaluated in PostgreSQL against a TPC-H-like workload. The experiment measures performance (*power*, *throughput*, and QphH) together with storage cost.

**Inputs:** schema metadata + SQL workload + prompt condition + ontology condition
**Step 1:** the LLM generates tuning recommendations (*indexes*, partial predicates, composite structures)
**Step 2:** recommendations are normalized and evaluated in PostgreSQL
**Step 3:** the TPC-H-like workload is executed and *power*, *throughput*, QphH, and storage are collected
**Output:** comparison across scenarios with and without ontology, with and without prompt, and across different model sizes.

**Figure 1. Overview of the experimental pipeline.**

### 3.2. Experimental factors

The evaluation combines four factors:

- **Model**: GPT-5.4, GPT-5.4-mini, GPT-5.4-nano, Gemini 2.5 Pro, Gemini 2.5 Flash, and Gemini 2.5 Flash-Lite;
- **Scale factor**: SF = 0.01, 0.02, 0.05, and 0.10;
- **Prompt**: with an explicit prompt versus without an explicit prompt;
- **Knowledge**: with ontology versus without ontology.

This yields 96 experimental scenarios, in addition to a baseline without extra tuning structures. The design enables separation of the effects of model capacity, task framing, and structured knowledge.

### 3.3. Role of the ontology

The ontology used in this work was originally proposed by Perciliano et al. [Perciliano et al. 2021]. In practice, it provides the model with a formalized vocabulary for reasoning about columns, tables, access patterns, and types of tuning actions. In the best-performing scenarios observed in this study, the ontology was associated with rules that favor:

- simple indexes for join keys and recurring filters;
- composite indexes when pairs of columns appear jointly in joins or filters; and
- partial indexes when a specific, reusable predicate appears consistently in the workload.

## 4. Experimental Methodology

The experiments were executed on PostgreSQL 14.22 on Ubuntu 22.04, with an Intel Core i5-1235U processor, 4 CPUs visible to the system, 5.8 GiB of RAM, and 1 TB of storage. The workload was TPC-H-like, with scale factors $SF = 0.01$, $SF = 0.02$, $SF = 0.05$, $SF = 0.1$ and four *streams*.

### 4.1. Evaluated structures

The structures generated in the experiments correspond exclusively to indexes, which fall into three main categories: simple indexes, composite indexes, and partial indexes.

### 4.2. Metrics

We use four main measures:

- ***Power***: measures system performance in isolated execution, being computed from the time required to execute the full set of workload queries and refresh functions sequentially. In general, the lower the total execution time, the higher the *Power*. Its formula can be expressed as:

$$Power = \frac{3600}{\text{Geometric Mean Time}} \times SF$$

  where *Geometric Mean Time* is the geometric mean of the sequential execution times of the workload queries and updates, and $SF$ is the benchmark scale factor.
- ***Throughput***: measures the system's ability under concurrent execution, considering multiple parallel streams running the same workload. Its computation relates the total number of operations executed to the total elapsed time in the parallel scenario, so that higher values indicate better simultaneous processing capacity. In general terms:

$$Throughput = \frac{\text{Total Number of Operations} \times 3600}{\text{Total Time in Seconds}} \times SF$$

  where the total number of operations depends on both the number of parallel streams and the workload per stream.
- **QphH**: is a composite metric that combines *Power* and *Throughput* through the geometric mean, aiming to jointly reflect system performance in both isolated and concurrent scenarios. Its formula is:

$$QphH = \sqrt{Power \times Throughput}$$

- **Tuning space (MB)**: corresponds to the total storage consumed by the structures recommended and effectively considered in the experiment, allowing the physical cost associated with performance gains to be analyzed. In general:

$$\text{Tuning space} = \sum_{i=1}^{n} \text{size of structure}_i$$

  where each structure corresponds to an index generated and materialized in the database.

In addition to these metrics, we use an auxiliary efficiency measure:

$$\text{Storage efficiency} = \frac{\text{QphH}}{\text{MB of additional structures}}.$$

This measure does not replace QphH, but it helps discuss the cost-benefit trade-off of the recommendations.

This research work enables us to explore the following and possibly other research questions, such as:

- **RQ1**: Does the ontology improve performance as measured by QphH?
- **RQ2**: Does the ontology improve the performance-to-space ratio?

- **RQ3**: Does the effect of the ontology depend on model capacity and on the use of a prompt?
- **RQ4**: To what extent can LLMs approximate DBA-like decision quality in relational index tuning?

All experimental material is available at: `https://github.com/EricRLeao1311/Test_LLM_Ontology`.

## 5. Results

Tables 1 and 2 present the consolidated results for all evaluated models (GPT and Gemini-based, respectively), conditions, and scale factors, in addition to the baseline without extra tuning structures. For each scale factor (*SF* = 0.01, 0.02, 0.05, and 0.10), the tables report *QphH*, tuning size in MB, and storage efficiency (*Eff.* = *QphH/MB*) for the four conditions: *P+O* (prompt + ontology), *P* (prompt only), *O* (ontology only), and *None* (neither prompt nor ontology). In total, the tables cover the baseline and 24 LLM-based configurations distributed across the evaluated models.

**Table 1. Results for GPT-based models across all conditions and scale factors. Each configuration reports QphH, tuning size (MB), and efficiency (Eff. = QphH/MB). Boldface indicates the best overall QphH and best overall Eff. for each scale factor.**

| Model | Cond. | $SF = 0.01$ | | | $SF = 0.02$ | | | $SF = 0.05$ | | | $SF = 0.10$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **QphH** | **MB** | **Eff.** | **QphH** | **MB** | **Eff.** | **QphH** | **MB** | **Eff.** | **QphH** | **MB** | **Eff.** |
| Baseline | – | 25.28 | 0.00 | – | 13.81 | 0.00 | – | 4.88 | 0.00 | – | 2.28 | 0.00 | – |
| 5.4 | P+O | 4551.60 | 2.01 | 2266.95 | 3442.25 | 3.86 | 891.92 | 2726.94 | 9.44 | **288.95** | 3286.47 | 18.73 | 175.42 |
| | P | 3320.22 | 4.19 | 792.89 | 2103.14 | 8.24 | 255.17 | 2430.17 | 20.34 | 119.46 | 3008.82 | 40.59 | 74.13 |
| | O | 3519.33 | 7.75 | 454.11 | 1566.09 | 14.65 | 106.91 | 1916.70 | 35.03 | 54.71 | 2516.98 | 68.98 | 36.49 |
| | None | 4475.04 | 11.73 | 381.36 | **5491.20** | 23.08 | 237.94 | **4536.95** | 56.93 | 79.69 | 2430.90 | 113.57 | 21.40 |
| 5.4 mini | P+O | 90.21 | 2.84 | 31.81 | 41.98 | 5.34 | 7.87 | 14.53 | 12.73 | 1.14 | 6.45 | 25.16 | 0.26 |
| | P | 1169.43 | 3.63 | 321.91 | 629.16 | 7.20 | 87.44 | 388.18 | 17.70 | 21.93 | 217.67 | 35.34 | 6.16 |
| | O | 1148.01 | 3.88 | 295.66 | 677.23 | 7.48 | 90.49 | 450.81 | 18.09 | 24.91 | 226.47 | 35.82 | 6.32 |
| | None | 537.02 | 3.41 | 157.66 | 372.61 | 6.52 | 57.19 | 234.68 | 15.88 | 14.78 | 225.15 | 31.77 | 7.09 |
| 5.4 nano | P+O | 63.79 | 3.67 | 17.37 | 35.03 | 7.27 | 4.82 | 12.78 | 17.94 | 0.71 | 5.62 | 35.89 | 0.16 |
| | P | 74.07 | 3.62 | 20.48 | 37.62 | 7.01 | 5.37 | 12.56 | 16.70 | 0.75 | 5.82 | 32.88 | 0.18 |
| | O | 871.27 | 8.62 | 101.11 | 846.70 | 16.48 | 51.39 | 3199.90 | 40.17 | 79.66 | 585.59 | 79.80 | 7.34 |
| | None | 603.11 | 6.52 | 92.45 | 755.74 | 12.53 | 60.31 | 634.73 | 30.05 | 21.12 | 353.14 | 59.50 | 5.94 |

*Abbreviations:* P+O, prompt + ontology; P, prompt only; O, ontology only; Eff., efficiency.

The central question of the paper is whether the ontology provides useful grounding for LLM-based tuning. The three measures used: QphH, storage footprint (MB), and storage efficiency (QphH/MB) make it possible to inspect the complete result set per scale factor, preserving the relationship between raw performance and physical cost.

Four high-level patterns emerge from the analysis of the Tables 1 and 2 together. First, the best raw QphH is not uniformly ontology-driven. At $SF = 0.01$, the best overall result is obtained by Gemini 2.5 Flash-Lite in the None condition, with QphH $= 5711.79$ and $7.95$ MB (Table 2). At $SF = 0.02$ and $SF = 0.05$, the best overall result is obtained by GPT-5.4 in the None condition, with QphH $= 5491.20$ and $4536.95$, respectively (Table 1). Only at $SF = 0.10$ does the ontology-driven condition achieve the best overall QphH: Gemini 2.5 Pro with P+O reaches QphH $= 4072.11$ with $16.84$ MB.

**Table 2. Results for Gemini 2.5-based models across all conditions and scale factors. Each configuration reports QphH, tuning size (MB), and efficiency (Eff. = QphH/MB). Boldface indicates the best overall QphH and best overall Eff. for each scale factor.**

| Model | Cond. | $SF = 0.01$ | | | $SF = 0.02$ | | | $SF = 0.05$ | | | $SF = 0.10$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **QphH** | **MB** | **Eff.** | **QphH** | **MB** | **Eff.** | **QphH** | **MB** | **Eff.** | **QphH** | **MB** | **Eff.** |
| Flash | P+O | 912.46 | 4.87 | 187.47 | 910.90 | 9.36 | 97.33 | 3756.31 | 22.52 | 166.83 | 2877.42 | 43.30 | 66.46 |
| | P | 882.26 | 6.02 | 146.47 | 3365.64 | 11.66 | 288.74 | 2292.42 | 28.56 | 80.26 | 3600.67 | 56.81 | 63.38 |
| | O | 2679.37 | 36.23 | 73.95 | 2926.53 | 70.52 | 41.50 | 2070.91 | 171.89 | 12.05 | 3361.51 | 339.27 | 9.91 |
| | None | 3316.12 | 29.34 | 113.01 | 3576.36 | 57.58 | 62.11 | 1730.17 | 141.84 | 12.20 | 809.37 | 283.27 | 2.86 |
| Flash-Lite | P+O | 967.12 | 11.15 | 86.75 | 957.63 | 21.54 | 44.46 | 3717.06 | 52.59 | 70.67 | 1888.38 | 104.37 | 18.09 |
| | P | 3664.95 | 16.97 | 215.98 | 2584.11 | 33.03 | 78.23 | 3612.65 | 81.21 | 44.48 | 1664.81 | 161.80 | 10.29 |
| | O | 1015.17 | 4.01 | 253.30 | 579.82 | 7.48 | 77.55 | 332.58 | 17.95 | 18.53 | 234.12 | 35.48 | 6.60 |
| | None | **5711.79** | 7.95 | 718.18 | 3407.90 | 15.27 | 223.13 | 2767.09 | 36.66 | 75.49 | 2021.35 | 72.15 | 28.02 |
| Pro | P+O | 4493.14 | 1.79 | **2511.45** | 3234.06 | 3.45 | **936.56** | 2450.35 | 8.48 | 288.81 | **4072.11** | 16.84 | **241.76** |
| | P | 985.61 | 2.28 | 432.05 | 966.02 | 4.45 | 216.93 | 2616.00 | 11.01 | 237.65 | 3306.97 | 21.90 | 151.01 |
| | O | 996.76 | 3.82 | 260.91 | 582.62 | 7.54 | 77.28 | 325.17 | 18.59 | 17.50 | 158.09 | 37.13 | 4.26 |
| | None | 85.57 | 3.81 | 22.45 | 42.84 | 7.50 | 5.71 | 18.54 | 18.45 | 1.00 | 9.01 | 36.82 | 0.24 |

*Abbreviations:* P+O, prompt + ontology; P, prompt only; O, ontology only; Eff., efficiency.

Second, although the absolute QphH winner is non-ontology-driven in the first three scale factors, the best ontology-driven configuration is consistently much more compact. At $SF = 0.01$, the best ontology-driven result is GPT-5.4 with P+O, reaching QphH $= 4551.60$ with only $2.01$ MB, which is $74.75\%$ less storage than the best non-ontology result at that scale. At $SF = 0.02$, GPT-5.4 with P+O reaches QphH $= 3442.25$ with $3.86$ MB, an $83.28\%$ reduction in extra storage relative to the best non-ontology result. At $SF = 0.05$, Gemini 2.5 Flash with P+O reaches QphH $= 3756.31$ with $22.52$ MB, still using $60.45\%$ less storage than the best non-ontology result. At $SF = 0.10$, the best ontology-driven configuration is also the best overall one, and still uses $70.35\%$ less storage than the best non-ontology competitor in that scale factor.

Third, the gains over the baseline remain extremely large even when the ontology is not the absolute raw-performance winner. The best ontology-driven QphH is about $180.02\times$ the baseline at $SF = 0.01$, $249.24\times$ at $SF = 0.02$, $770.31\times$ at $SF = 0.05$, and $1787.79\times$ at $SF = 0.10$. In other words, ontology-driven grounding is never associated with a marginal result in this dataset: even in the scales where it does not produce the single largest QphH, it still produces very strong and practically relevant tuning outcomes.

Fourth, the effect of ontology grounding is strongly model-dependent. In larger models, the combination of prompt and ontology tends to be highly beneficial, whereas in smaller models, the same combination often degrades performance, suggesting that the additional semantic structure increases task complexity beyond what weaker models can reliably integrate.

### 5.1. Recommendation profile

**Table 3. Profile of the generated recommendations.**

| Scenario | Total | Simple | Composite | Partial |
|---|---|---|---|---|
| P+O / GPT-5.4 | 5 | 2 | 2 | 1 |
| P / GPT-5.4 | 9 | 3 | 6 | 0 |
| O / GPT-5.4 | 21 | 8 | 9 | 4 |
| None / GPT-5.4 | 24 | 5 | 16 | 3 |
| P+O / Mini | 8 | 2 | 6 | 0 |
| P / Mini | 7 | 0 | 7 | 0 |
| O / Mini | 11 | 0 | 8 | 3 |
| None / Mini | 10 | 0 | 7 | 3 |
| P+O / Nano | 4 | 0 | 3 | 1 |
| P / Nano | 4 | 0 | 4 | 0 |
| O / Nano | 26 | 24 | 0 | 2 |
| None / Nano | 15 | 1 | 13 | 1 |
| P+O / Gemini 2.5 Flash | 20 | 13 | 4 | 3 |
| P / Gemini 2.5 Flash | 23 | 13 | 10 | 0 |
| O / Gemini 2.5 Flash | 109 | 36 | 33 | 40 |
| None / Gemini 2.5 Flash | 49 | 27 | 21 | 1 |
| P+O / Gemini 2.5 Flash-Lite | 31 | 13 | 15 | 3 |
| P / Gemini 2.5 Flash-Lite | 42 | 20 | 20 | 2 |
| O / Gemini 2.5 Flash-Lite | 23 | 17 | 2 | 4 |
| None / Gemini 2.5 Flash-Lite | 20 | 4 | 14 | 2 |
| P+O / Gemini 2.5 Pro | 6 | 3 | 2 | 1 |
| P / Gemini 2.5 Pro | 6 | 3 | 2 | 1 |
| O / Gemini 2.5 Pro | 6 | 2 | 2 | 2 |
| None / Gemini 2.5 Pro | 6 | 1 | 4 | 1 |

Table 3 shows a large variation in recommendation volume across scenarios, from only 4 structures in *P+O / Nano* and *P / Nano* to 109 in *O / Gemini 2.5 Flash*. Intermediate cases also differ substantially: the GPT-5.4 family ranges from 5 to 24 recommendations, the Mini family from 7 to 11, the Nano family from 4 to 26, and the Gemini families from compact sets of 6 indexes in all *Gemini 2.5 Pro* settings to much larger sets in *Gemini 2.5 Flash* and *Gemini 2.5 Flash-Lite*. Quantitatively, this indicates that the scenarios differ not only in performance, but also in how broadly they explore the space of candidate structures.

The distribution by index type also reveals distinct recommendation styles. Composite indexes dominate many scenarios, such as *P / GPT-5.4*, *P+O / Mini*, *P / Mini*, *None / GPT-5.4*, and *None / Nano*, suggesting a preference for access paths that combine join and filter attributes. In contrast, *O / Nano* is almost entirely driven by simple indexes, with 24 out of 26 recommendations in this category, while *O / Gemini 2.5 Flash* combines high volume with high heterogeneity, including 36 simple, 33 composite, and 40 partial indexes. The *Gemini 2.5 Pro* cases are the most stable, always producing 6 structures with only small changes in composition.

Partial indexes appear unevenly across the scenarios. In some cases, they are absent, such as *P / GPT-5.4*, *P+O / Mini*, *P / Mini*, *P / Nano*, and *P / Gemini 2.5 Flash*; in others, they play a more visible role, especially in *O / Gemini 2.5 Flash*, where they ac-

count for 40 recommendations. This suggests two distinct behaviors: some settings favor more general-purpose structures, whereas others are more willing to encode workload-specific predicates directly into the recommended design. Overall, the table shows that prompt, ontology, and model size affect not only how many structures are generated but also the type of physical design logic favored.

A qualitative look at the indices themselves reinforces this difference. Some recommendations appear repeatedly across many scenarios, especially indexes on `lineitem(l_orderkey)`, `orders(o_custkey)` or `orders(o_custkey, o_orderdate)`, `partsupp(ps_partkey, ps_suppkey)`, and `lineitem(l_partkey, l_suppkey)`, which indicates a shared emphasis on the main join paths of the workload. At the same time, a few scenarios introduce much more specialized structures, such as partial indexes on `o_orderstatus = 'F'`, `l_returnflag = 'R'`, `c_phone` prefixes, `PROMO`-type filters, or highly specific predicates over brand, container, ship mode, and region values. There are also signs of redundancy in some larger recommendation sets, especially when a scenario suggests both simple and composite variants over the same leading columns, or multiple composites with very similar prefixes, which is particularly visible in the larger Gemini outputs. This suggests that larger recommendation sets are not necessarily more refined: they often capture more workload details, but they may also include overlapping or overly specialized structures.

### 5.2. Storage efficiency and compactness

**Table 4. Best storage-efficiency results with and without ontology across scale factors.**

| SF | Best w/ ontology | Best w/o ontology | Δ Eff. |
|---|---|---|---|
| 0.01 | Gemini 2.5 Pro (P+O) | GPT-5.4 (P) | +216.75% |
| 0.02 | Gemini 2.5 Pro (P+O) | Gemini 2.5 Flash (P) | +224.36% |
| 0.05 | GPT-5.4 (P+O) | Gemini 2.5 Pro (P) | +21.59% |
| 0.10 | Gemini 2.5 Pro (P+O) | Gemini 2.5 Pro (P) | +60.09% |

The clearest pattern in favor of ontology-driven grounding emerges when storage cost is explicitly incorporated into the analysis. As shown in Table 4, the highest storage-efficiency result at every evaluated scale factor is achieved by an ontology-driven configuration. At $SF = 0.01$, Gemini 2.5 Pro with P+O reaches QphH/MB $= 2511.45$, with QphH $= 4493.14$ and only $1.79$ MB of recommended structures. The same configuration remains the most efficient at $SF = 0.02$, reaching QphH/MB $= 936.56$. At $SF = 0.05$, the best ratio is obtained by GPT-5.4 with P+O, with QphH/MB $= 288.95$. At $SF = 0.10$, Gemini 2.5 Pro with P+O again leads, with QphH/MB $= 241.76$.

This advantage remains clear when the best ontology-driven configuration is compared with the best configuration that does not use ontology. The gains in storage efficiency are substantial at all scales: $+216.75\%$ at $SF = 0.01$, $+224.36\%$ at $SF = 0.02$, $+21.59\%$ at $SF = 0.05$, and $+60.09\%$ at $SF = 0.10$. Importantly, these gains are not explained by larger physical designs. On the contrary, the ontology-driven winners are also compact, using only $1.79$, $3.45$, $9.44$, and $16.84$ MB, respectively. In three of the four

scale factors, the ontology-driven winner also delivers higher raw QphH than the best no-ontology alternative; at $SF = 0.02$, its QphH is slightly lower, but this difference is more than compensated by a much smaller storage footprint, yielding the highest overall cost-performance ratio.

Taken together, these results provide the strongest evidence for the ontology as a grounding mechanism. For **RQ2**, the answer is clearly positive: once storage is treated as a first-class evaluation criterion, ontology-driven configurations dominate the performance-to-space frontier in all four scale factors. This pattern also refines the answers to **RQ1** and **RQ3**: the ontology does not uniformly maximize raw QphH across all models and conditions, but it consistently improves the trade-off between performance and footprint, with effects that vary according to model capacity and prompt use. This behavior is consistent with grounding, since the ontology appears to steer the models toward more compact and better-targeted recommendations rather than merely adding more information. For **RQ4**, the results suggest that LLMs can approximate DBA-like decision quality only to a limited extent: while some strong configurations produce useful recommendations, their behavior is not stable enough to support autonomous DBA replacement. The consistent gains in storage efficiency further indicate that LLMs alone do not reliably achieve the same compactness and cost-effectiveness without external structured guidance.

## 6. Discussion

### 6.1. Why does the ontology help in the best case?

The results indicate that the ontology's benefit is not limited to a single, isolated configuration. Instead, the strongest outcomes across scale factors tend to occur when the ontology is combined with prompt guidance in the most capable models. This suggests that, in the best cases, the ontology serves as a semantic grounding mechanism, making the recommendations more precise and better aligned with the workload.

Rather than simply expanding the set of candidate structures, the ontology appears to help the model select structures more precisely, reducing redundancies and avoiding low-impact recommendations. This interpretation is reinforced by the fact that the best-performing ontology-driven configurations are also among the most storage-efficient, indicating that the gain is not just due to recommending more indexes but to recommending better ones.

Therefore, the main contribution of the ontology, in the best cases, seems to be its ability to guide the model toward structures that are both effective and economically optimal. In this sense, the ontology does not function merely as additional context, but as a semantic constraint that improves recommendation quality.

### 6.2. Why do prompt and ontology + prompt make things harder for smaller models?

In the smaller models, the results suggest that combining prompt and ontology may increase task complexity. Unlike the strongest models, these variants appear to have greater difficulty in stably integrating textual instructions, workload evidence, schema information, and semantic relationships. As a consequence, the additional guidance does not

always function as support and may instead reduce the coherence of the recommendations.

At the same time, the evidence suggests a more nuanced interpretation than simply assuming that ontology is harmful in weaker models. The difficulty seems to arise less from the ontology itself and more from the model's limited ability to absorb and prioritize multiple sources of guidance simultaneously. In this setting, prompt engineering and ontology can cease to be complementary and start competing for the model's limited inferential capacity.

This indicates that for smaller models, more context does not necessarily lead to better decisions. When reasoning capacity is constrained, the accumulation of instructions and structured knowledge may make it harder to identify the truly relevant workload signals, which helps explain the instability observed in weaker configurations.

### 6.3. Implications for DBA support systems

For DBA support systems, the results reinforce the idea that ontologies can be useful, but their incorporation should not be applied uniformly. Their value depends on model capacity, on the way the knowledge is presented, and on whether the system is optimizing only raw performance or also structural economy. In stronger models, the ontology can substantially improve recommendation quality; in weaker ones, it may be preferable to translate ontology knowledge into simpler intermediate heuristics or rules.

More broadly, the results reinforce that tuning-support tools should evaluate not only QphH in isolation, but also storage efficiency, structural compactness, and workload alignment. Some of the most interesting ontology-driven outcomes are precisely those that remain on the efficiency frontier, showing that better semantic grounding can improve not only effectiveness but also the cost-benefit profile of the recommendations.

In this context, the ontology remains promising, but its integration with LLMs should be treated as part of a carefully designed architecture rather than as an automatic addition of context. The practical implication is that DBA support systems may benefit from adaptive strategies, in which the form of ontology use depends on the reasoning capacity of the selected model.

### 6.4. When does the ontology help the LLM?

The results suggest that the ontology helps the LLM in two main situations. The first is when the model is sufficiently robust to exploit both semantic structure and prompt guidance simultaneously. In these cases, the best results tend to emerge in $P{+}O$ configurations, especially for the strongest models, indicating that the ontology works as a precision layer that improves structure selection rather than merely adding more context.

The second situation is when the objective is not limited to faster query performance but also includes broader database concerns such as maintainability, scalability, and structural economy. In these cases, ontology-driven knowledge may be useful because it can guide the model toward recommendations that are not only effective for the workload but also more selective and less redundant. This is particularly important in tuning scenarios where the cost of additional structures matters, since every extra index may increase storage consumption and maintenance overhead during updates, insertions, and deletions.

Taken together, these patterns indicate that the ontology is not beneficial for a single reason. In stronger models, it helps by refining and constraining an already capable reasoning process; in broader tuning scenarios, it helps by steering the model toward recommendations that better balance performance with structural cost. This also helps explain why the ontology should not be interpreted as a uniformly additive resource: its contribution is comparative, model-dependent, and closely tied to how the semantic knowledge is integrated into the recommendation process.

## 7. Conclusions

This paper investigated whether a domain ontology can improve database tuning recommendations generated by Large Language Models. Overall, the results show that the ontology has a positive effect, but one that is strongly dependent on model capacity and on the way the task is framed. In the strongest configurations, the combination of the prompt and the ontology yielded the best outcomes, indicating that structured domain knowledge can serve as a semantic grounding mechanism to improve recommendation quality.

At the same time, the results do not support the idea that LLMs can replace the DBA in relational tuning tasks. Although some configurations achieved strong raw performance, the models' overall behavior was not sufficiently stable or consistently efficient across conditions to justify autonomous replacement. In this sense, the importance of the ontology lies precisely in showing that LLMs do not reliably reproduce DBA-like tuning decisions on their own: structured knowledge is often necessary to constrain the model and steer it toward more compact, selective, and cost-effective recommendations.

From a practical perspective, these findings suggest that ontologies are a promising component for DBA support systems, especially when tuning must balance performance, structural economy, maintainability, and long-term scalability.

As future work, this study can be extended in five main directions: evaluating additional LLMs, including local and open-weight models; comparing ontology-guided LLM recommendations against traditional index advisors and DBMS tuning tools; extending the analysis beyond indexes to other physical design actions, especially materialized views; replicating the experiments with larger scale factors and real, dynamic workloads beyond the TPC-H-like setting used in this paper; and analyzing the generated index sets more deeply in terms of redundancy, selectivity, overlap, and workload alignment.

## 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. Eric Ruas Leão 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

Alencar, N., Brayner, A., Monteiro, J. M., and de Aguiar Moraes Filho, J. (2019). Dac-join: A join operator for improving database performance on modern hardware. Concurr. Comput. Pract. Exp., 31(17).

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

Almeida, A. C., Campos, M. L. M., Baião, F., Lifschitz, S., de Oliveira, R. P., and Schwabe, D. (2019). An ontological perspective for database tuning heuristics. In Laender, A. H. F., Pernici, B., Lim, E.-P., and de Oliveira, J. P. M., editors, Conceptual Modeling, pages 240–254, Cham. Springer International Publishing.

Calero, C., Ruiz, F., Baroni, A., e Abreu, F. B., and Piattini, M. (2006). An ontological approach to describe the sql: 2003 object-relational features. Computer Standards & Interfaces, 28(6):695–713.

de Aguiar, C. Z., Falbo, R. A., and Souza, V. E. S. (2018). Ontological Representation of Relational Databases. In Proc. of the 11th Seminar on Ontology Research in Brazil (ONTOBRAS 2018), pages 140–151, São Paulo, SP, Brazil. CEUR.

de Araújo, A. H. M., Monteiro, J. M., de Macêdo, J. A. F., Tavares, J. A., Brayner, A., and Lifschitz, S. (2014). On using an online, automatic and non-intrusive approach for rewriting SQL queries. J. Inf. Data Manag., 5(1):28–39.

Dou, H., Jin, L., Zhou, Y., He, J., Zhang, Y., and Zheng, Z. (2026). Demotuner: Automatic performance tuning for database management systems based on demonstration reinforcement learning.

Fuentes, A. D., Almeida, A. C., de Carvalho Costa, R. L., Braganholo, V., and Lifschitz, S. (2018). Database tuning with partial indexes. In Lóscio, B. F., Dorneles, C. F., and Barioni, M. C. N., editors, XXXIII Simpósio Brasileiro de Banco de Dados, SBBD 2018, Rio de Janeiro, RJ, Brazil, August 25-26, 2018, pages 181–192. SBC.

Giannakouris, V. and Trummer, I. (2025). $\lambda$-tune: Harnessing large language models for automated database system tuning. Proc. ACM Manag. Data, 3(1).

Lao, J., Wang, Y., Li, Y., Wang, J., Zhang, Y., Cheng, Z., Chen, W., Tang, M., and Wang, J. (2025). Gptuner: An llm-based database tuning system. ACM SIGMOD Record, 54(1):101–110.

Li, Y., Li, H., Pu, Z., Zhang, J., Zhang, X., Ji, T., Sun, L., Li, C., and Chen, H. (2024). Is large language model good at database knob tuning? a comprehensive experimental evaluation. arXiv preprint arXiv:2408.02213.

Li, Y., Li, H., Zhang, J., Borovica-Gajic, R., Wang, S., Zhang, T., Chen, J., Shi, R., Li, C., and Chen, H. (2025). Agenttune: An agent-based large language model framework for database knob tuning. Proc. ACM Manag. Data, 3(6):1–29.

Liu, F., Wei, R., Wang, H., Ding, Z., Mo, Z., Zhou, K., and Liu, Y. (2026). Arbiter: Towards joint and fine-grained index and partition tuning in analytical databases. Inf. Process. Manag., 63(5).

Metamodel, C. W. (2003). Common warehouse metamodel (cwm) specification. Inc., Needham, MA, USA.

Mozaffari, M., Dignos, A., Gamper, J., and Störl, U. (2024). Self-tuning database systems: A systematic literature review of automatic database schema design and tuning. ACM Computing Surveys, 56(11).

Oliveira, R. P., Baião, F., Machado, J., Almeida, A. C., and Lifschitz, S. (2022). Autonomic combination and selection of tuning actions. In Anais do XXXVII Simpósio Brasileiro de Bancos de Dados (SBBD), pages 39–51.

Ouared, A., Ouhammou, Y., and Roukh, A. (2016). A meta-advisor repository for database physical design. In International Conference on Model and Data Engineering, pages 72–87. Springer.

Perciliano, L. S. S., dos Santos, V., Baião, F., Haeusler, E. H., Lifschitz, S., and Almeida, A. C. (2021). Inferencing relational database tuning actions with ondbtuning ontology. In Anais do XXXVI Simpósio Brasileiro de Bancos de Dados (SBBD), pages 157–168.

Souza, V. A. L. L. and Lifschitz, S. (2022). Tuningchef: an approach for choosing the best cost-benefit database tuning actions. In Anais do XXXVII Simpósio Brasileiro de Bancos de Dados (SBBD).

Wang, X., Wu, W., Narasayya, V., and Chaudhuri, S. (2026). Evaluating the practical effectiveness of llm-driven index tuning with microsoft database tuning advisor.

Wu, Y., Zhou, X., Zhang, Y., and Li, G. (2024). Automatic index tuning: A survey. IEEE Transactions on Knowledge and Data Engineering, 36(12):7657–7676.

Xue, S., Jiang, C., Shi, W., Cheng, F., Chen, K., Yang, H., Zhang, Z., He, J., Zhang, H., Wei, G., Zhao, W., Zhou, F., Qi, D., Yi, H., and Liu, S. (2024). Db-gpt: Large language model meets database. Data Science and Engineering, 9:1–16.

Zhao, X., Zhou, X., and Li, G. (2023). Automatic database knob tuning: A survey. IEEE Transactions on Knowledge and Data Engineering, 35(12):12470–12490.