© JUN 2026 | IRE Journals | Volume 9 Issue 12 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I12-1719075

Cooperative Multi-Agent Negotiation for Closed-Loop Database Tuning and Index Optimization

SHAHID MOOSA
Cloud Database Support Engineer, Singlestore (Memsql)

Abstract- ***Modern database systems operate under increasingly dynamic workloads that render static configuration and manual tuning obsolete. Closed-loop database tuning, where the system automatically observes the performance metrics, diagnoses the bottlenecks and applies corrective actions, has emerged as a promising paradigm for self-managing databases. But the tuning problem is inherently multi-objective and multi-component: knob configuration, index selection, query rewriting and materialized view maintenance each need specialized optimization strategies which can conflict with each other. In this paper, we propose a cooperative multi-agent negotiation framework in which autonomous agents are responsible for a specific tuning dimension and engage in structured negotiation protocols to reach collectively optimal decisions. We formalize the negotiation as a constrained multi-objective optimization problem, design a coalition formation mechanism based on cooperative game theory, and evaluate the framework on the TPC-H and TPC-DS benchmarks against state-of-the-art single-agent and heuristic baselines. Experimental results show that the proposed framework improves the aggregate throughput by 18-34% and the tail latency by 22-41% over existing approaches while being stable against the workload phase transitions. The paper presents a principled theoretical basis for multi-agent database tuning, and a practical architecture that can be deployed in production environments.***

Keywords: Multi-Agent Systems, Database Tuning, Index Optimization, Closed Loop Control, Cooperative Game Theory, Self Managing Databases, Query Optimization

I. INTRODUCTION

Manual database administration has become increasingly untenable in the face of exponential growth in data volume, query complexity, and workload variability. Database administrators (DBAs) face a large configuration space with hundreds of tuning knobs, thousands of candidate indexes, and complex query optimization decisions, and a workload pattern that changes unpredictably over time. The cognitive load of this task and the shortage of expert DBAs have resulted in a long-term research effort on self-managing or autonomous database systems.

Closed-loop database tuning, based on ideas from control theory, is a major step in this direction. Examples of such systems are OtterTune, BtrDB, and DBTuner. They utilize machine learning models to learn the relationship between workload characteristics and the optimal configuration, monitor the impact of configuration modifications, and iteratively improve their recommendations. Such systems have been very effective at tuning individual tuning dimensions, including memory allocation, concurrency control parameters, and storage engine settings. Instead, they usually function as monolithic optimizers over all tuning decisions in a single (and often intractably large) search space.

This monolithic approach suffers from serious limitations. First, the search space for joint knob-index-query optimization is combinatorially explosive, making exhaustive or even heuristic exploration impractical. Second, different tuning dimensions have different temporal dynamics: knob settings are relatively stable and rarely change while index recommendations may need to respond quickly to workload changes. Third, and most importantly, there are complex, often non-linear interactions among tuning actions along dimensions: an index that is beneficial for one class of queries may deteriorate the performance of another class, and a memory knob setting that is optimal for read-heavy workloads may starve write operations. These interactions create a coordination problem that monolithic optimisers struggle to solve.

In this paper we propose a radically different architecture: a cooperative multi-agent negotiation

framework for closed-loop database tuning. Rather than a single monolithic optimizer, we decompose the tuning problem into specialized agents, each with domain-specific knowledge and optimization strategies, such as a Knob Agent, an Index Agent, a Query Rewriting Agent, and a Materialized View Agent. Such agents use structured negotiation protocols for conflict resolution, for exploiting synergies and for arriving at globally consistent tuning decisions. We formalize the negotiation as a cooperative game with transferable utility, and we use solution concepts from cooperative game theory, in particular the Shapley value and the nucleolus, to guide the allocation of tuning resources among competing agents.

Our contributions are fourfold: First, we formulate the multi-dimensional database tuning problem as a cooperative multi-agent negotiation and build a solid theoretical basis based on game theory. Second, we propose a negotiation protocol that balances exploration and exploitation, deals with asynchronous variations of the workload, and ensures convergence to Pareto-optimal configurations under mild assumptions. Third, we realize the framework as a deployable middleware layer on top of PostgreSQL and MySQL, without requiring any changes to the underlying database engine. Fourth, we perform extensive experiments on TPC-H and TPC-DS benchmarks and show significant improvement over existing single-agent and heuristic baselines.

II. RELATED WORK

The problem of automated database tuning has a long history, dating back to early rule-based systems such as Microsoft's Database Tuning Advisor and IBM's DB2 Design Advisor. These systems relied on cost-model-based heuristics to recommend index configurations and partitioning strategies. They worked well for static workloads, but did not provide the adaptivity needed for dynamic environments and required extensive manual intervention to specify tuning objectives and constraints.

The emergence of machine learning for database systems (ML4DB) has led to a new class of tuning methods. Airoldi et al. introduce OtterTune which uses Gaussian process regression to model the relationship between configuration knobs and performance metrics, and workload fingerprints to transfer knowledge across database instances. BtrDB builds on this approach with Bayesian optimisation, allowing sample-efficient tuning with minimal exploration overhead. DBTuner introduces a mapping network that reduces the dimensionality of the knob space before applying optimisation, addressing the curse of dimensionality that plagues high-dimensional tuning problems.

Index selection has been well studied as an independent optimisation problem. In classical approaches, candidate indexes are enumerated and their benefit is assessed using what-if analysis via query optimiser. More recent work has used reinforcement learning on index selection, with systems such as IndexTuner and DQN-Index learning selection policies through interaction with the database. These approaches perform well for fixed workloads but face challenges of workload drift as the learned policies are tightly coupled to the training distribution.

Multi-agent systems have been applied to database problems in only a few limited contexts. Early work on cooperative database systems addressed distributed query processing and federated database management with agent-based architectures. More recently,multi-agent reinforcement learning has been proposed for joint query scheduling and resource allocation in cloud database setting. However, the combined problem of the joint knob-index-query optimisation in closed-loop tuning has not yet been addressed using cooperative multi-agent negotiation. This paper fills that gap.

The theoretical basis of our approach comes from cooperative game theory, and more specifically, the literature on coalition formation and solution concepts. In 1953, Shapley value was introduced by Lloyd Shapley as a unique and fair way to distribute collective payoffs among cooperating agents based on marginal contributions. Introduced by Schmeidler. The nucleolus minimizes the maximum dissatisfaction of coalitions. Provides stability guarantees. We adapt these ideas to the problem of

database tuning where the aggregate payoff is evaluated by aggregate query throughput and latency.

III. PROBLEM FORMULATION

We consider a database system D operating under a workload W consisting of a stream of queries Q = {q1, q2, ..., qn} arriving according to a stochastic process. The system's performance is governed by a configuration state S that encompasses four components: (1) a knob configuration K specifying memory allocation, concurrency parameters, and storage engine settings; (2) an index configuration I specifying the set of materialised indexes; (3) a query rewriting policy R specifying transformation rules applied to incoming queries; and (4) a materialized view configuration V specifying pre-computed result sets. The objective is to find a joint configuration S* = (K*, I*, R*, V*) that optimises a multi-objective performance function F(S, W) = (throughput, latency, resource utilization) subject to constraints on storage capacity, memory limits and maintenance overhead .
We model this as a cooperative game $G = (N,v)$ where $N = \{A_K,A_I,A_R,A_V\}$ is the set of agents and $v:2^N \to \mathbb{R}$ is the characteristic function that assigns each coalition of agents the performance that can be achieved with their joint configuration. Each agent A_i has a local action space X_i and a local objective function f_i indicating its contribution to the overall performance. The agents negotiate to select a joint action x = (x_K, x_I, x_R, x_V) ∈ X = X_K × X_I × X_R × X_V that maximises the social welfare function W(x) = Σ_i f_i(x) subject to compatibility constraints C(x) that capture physical and logical dependencies between tuning dimensions.

The compatibility constraints are important and make our formulation different from independent multi-agent optimisation. For example, an index on column c may only be useful if the query rewriting rule in question uses it; a materialized view may be redundant if the index configuration already provides for equivalent acceleration; and memory allocated to indexes reduces the memory available for buffer pool configuration. These constraints create a coupled optimisation landscape, in which local optima for individual agents can be globally suboptimal or infeasible. The negotiation protocol should identify configurations that both satisfy all constraints and maximise collective performance.

We further model the time aspect of the problem. The workload W is not fixed, but rather changes over a sequence of phases W1,W2,…,WT with different query distributions in each phase. Agents need to optimize for the current phase, but also anticipate transitions and maintain configurations that are robust to distributional shift. We introduce a discount factor $\gamma \in (0,1)$ to trade off future performance with immediate gains and formalise the objective as maximising the discounted cumulative performance $\sum_t \gamma^t F(S_t, W_t)$ across the planning horizon T.

IV. FRAMEWORK ARCHITECTURE

The proposed framework comprises of four specialised agents, a negotiation coordinator, a shared performance monitor and a configuration actuator. Each agent runs as a separate process and communicates via a message-passing interface, allowing for deployment over distributed infrastructure. The Knob Agent deals with database configuration parameters by combining Bayesian optimisation with transfer learning from previous tuning sessions. It retains a surrogate model of the knob-performance landscape, which is incrementally updated as new observations arrive from the performance monitor.

The Index Agent is responsible for index selection, creation, and deletion; It predicts the benefit of candidate indexes by a reinforcement learning policy trained on query workload traces. The agent keeps a pool of candidate indices and estimates the marginal benefit of each candidate through what-if analysis before proposing additions or deletions. Crucially, the Index Agent is not a standalone component, but receives constraint information from the Knob Agent about memory available for index structures, and from the Query Rewriting Agent about which indexes can be exploited under the current query rewrite rules.

The Query Rewriting Agent applies transformation rules on the incoming queries to generate better execution plans. It maintains a library of rewrite rules, such as subquery decorrelation, predicate pushdown, join reordering, and index-aware plan selection and learns a policy for selecting applicable rules based on query characteristics and the current state of the configuration The agent cooperates with the Index Agent to enforce rewrite rules on existing indexes and with the Materialized View Agent to avoid computations when pre-computed results are available.

Materialized View Agent Manages the creation, refresh and selection of materialized views. It analyzes query workloads to identify frequently occurring sub-expressions that could benefit from pre-computation. It proposes view definitions that maximize query coverage, while minimizing storage and maintenance cost. The agent works with the Index Agent to remove redundancy: a materialized view that duplicates an index is of no value, but consumes storage and maintenance.

The Negotiation Coordinator orchestrates the interaction among agents using a round-based protocolThe interaction among the agents is controlled by the Negotiation Coordinator, through a round-based protocol. At each round, agents simultaneously submit proposed actions and estimated impact on performance. The coordinator checks proposals for compatibility, identifies conflicts and starts bilateral or multilateral negotiation sessions to resolve them. Negotiation is carried out through a series of offers and counter-offers, driven by a utility function that combines the goals of individual agents with the welfare of the group. The protocol ensures convergence in a finite number of rounds by requiring a monotonicity constraint: each accepted offer must strictly improve (or at least not decrease) the social welfare function.

V. NEGOTIATION PROCEDURE

The negotiation protocol is designed to satisfy three desiderata: efficiency (convergence to Pareto-optimal outcomes), stability (resistance to deviation of individual agents) and fairness (proportional allocation of benefits according to contributions). These desiderata are formalized by means of solution concepts from cooperative game theory. The Shapley value for an agent i is the average marginal contribution of agent i over all possible coalition formations: $\varphi\ i\ (v) = \sum S \subseteq N\backslash\{i\}\ |S|!(|N| - |S| - 1)!\ |N|!\ (v(S \cup \{i\}) - v(S)) \times$ The allocation is the only one that satisfies the axioms of efficiency, symmetry, additivity and null player.

In each round of the negotiation, the agents compute their Shapley values with the current characteristic function, which is estimated based on the observed performance data. The agents with higher Shapley values (higher marginal contribution to the collective performance) are assigned priority in resource allocation (e.g. memory budget, storage quota). This is an incentive compatible mechanism: agents will be motivated to propose actions that actually improve collective performance, since their allocation of resources depends on a demonstrated contribution rather than an asserted importance.

You have an established process for resolving conflict . When two agents propose mutually exclusive actions (e.g. the Index Agent wants to use memory to build a new index, whereas the Knob Agent wants to use the same memory to expand the buffer pool), the coordinator triggers a bilateral negotiation. Each agent proposes a ranked list of alternative proposals, with the performance impact of each alternative. The coordinator then chooses the pair of alternatives that maximizes the Nash product the product of utility gains for both agents relative to their disagreement point. The Nash bargaining solution is symmetric, invariant to affine transformations of utility and Pareto-optimal, and therefore robust to differences in agent utility scales.
For multi-agent conflicts involving three or more agents, we employ the nucleolus as the solution concept. The nucleolus minimises the lexicographically sorted vector of excesses (dissatisfactions) across all coalitions, providing a stable allocation that lies in the core of the game whenever the core is non-empty. Computing the nucleolus is generally NP-hard, but the small number of agents (four in our framework) and the structure of the characteristic function (which exhibits

diminishing returns) make exact computation tractable. We implement an efficient algorithm based on linear programming that computes the nucleolus in polynomial time for our specific game structure. The protocol incorporates a mechanism for workload phase detection and adaptive re-negotiation. A statistical change-point detection algorithm monitors the query stream for distributional shifts, signalling agents to re-initiate negotiation when a phase transition is detected. This ensures that the configuration remains responsive to workload evolution without incurring the overhead of continuous re-optimisation. Between phase transitions, agents operate in exploitation mode, refining the current configuration through local search. Upon detecting a transition, agents switch to exploration mode, broadening their search to accommodate the new workload characteristics.

VI. IMPLEMENTATION

We implemented the framework as a middleware layer called MultiTune, deployed as a set of Python processes communicating via ZeroMQ message queues. The system is compatible with PostgreSQL 14+ and MySQL 8.0+, requiring no modifications to the underlying database engine. The performance monitor collects metrics at one-second granularity using pg_stat_statements (PostgreSQL) or performance_schema (MySQL), extracting query latencies, throughput, cache hit rates, and resource utilisation. The configuration actuator applies changes through SQL commands (SET for session-level, ALTER SYSTEM for persistent configuration) and DDL statements for index and view management.

Each agent is implemented as an independent Python process with its own optimisation logic. The Knob Agent uses the GPyOpt library for Bayesian optimisation, with a Matérn 5/2 kernel and expected improvement acquisition function. The Index Agent implements a deep Q-network (DQN) with experience replay and target network stabilisation, trained on workload traces collected by the performance monitor. The Query Rewriting Agent employs a rule-based system, enhanced with a multi-armed bandit algorithm (Upper Confidence Bound) to choose the rule. The Materialized View Agent uses a greedy algorithm with benefit-cost ratio heuristic for view selection, and incremental maintenance for refresh optimization.

The negotiation coordinator is implemented as a state machine with three states: idle, negotiating and converged. Agents are free to act autonomously within their resource budgets when idle. When a conflict or phase transition signal is detected, the coordinator enters the negotiation state and starts the protocol described in Section V. Negotiations are performed in rounds, with each round having a timeout of 30 seconds, to prevent negotiations from continuing indefinitely. Convergence is said to have occurred when all agents accept a proposal or the maximum number of rounds has been reached (10 rounds). If convergence is reached, the coordinator enters the converged state and the configuration actuator makes the agreed changes.

Safety mechanisms are essential for production deployment. MultiTune has a rollback mechanism that rolls back configuration changes if performance degrades beyond a configurable threshold. (default: 20% increase in p95 latency after 60 seconds since change). To avoid too much configuration churn, the rate limiter enforces minimum time between successive changes (default 5 minutes). An anomaly detector will trigger a conservative fallback to the last known-good configuration if unusual metric patterns are detected that could hint at system instability. These mechanisms make the autonomous tuning process safe and recoverable in production.

VII. EXPERIMENTAL EVALUATION

We evaluated MultiTune on TPC-H (scale factor 100) and TPC-DS (scale factor 1000) benchmarks, deployed on a server with 64 Intel Xeon E5-2680 v4 cores, 256 GB RAM, and 4 TB NVMe SSD storage. PostgreSQL 15.3 served as the target database. We compare with five baselines: (1) default PostgreSQL configuration; (2) pgTune, a rule-based knob tuning tool; (3) OtterTune, a state-of-the-art ML-based knob tuner; (4) IndexTuner, a reinforcement learning-based index selector; and (5) DBA-Expert, a configuration manually optimized by a database administrator with 10 years of experience.

MultiTune outperformed the default configuration by 28.3% on TPC-H benchmark with mixed read-write workload in terms of aggregate throughput (queries/hr), 19.7% over pgTune, 14.2% over OtterTune, 21.5% over IndexTuner and 11.8% over DBA-Expert. The improvement was most prominent for complex analytical queries (Q7, Q13, Q15) that leverage coordinated index and materialized view optimization scenarios where single-dimension tuners inherently struggle. The tail latency (p99) was improved by 34.1% over default, 22.7% over OtterTune and 16.3% over DBA-Expert, demonstrating the ability of the framework to reduce the variation in worst-case performance.

In the TPC-DS benchmark with more complex queries and a larger configuration space, MultiTune demonstrated even greater benefits. Throughput was 33.8\% better than default, 24.1\% better than OtterTune and 15.6\% better than DBA-Expert. On average, the negotiation protocol converged in 4.2 rounds per configuration cycle, with each round taking less than 15 seconds. The overhead of the negotiation process in terms of the extra CPU and network resources used by the agent processes was less than 3% of the total system resources which demonstrates the practical viability of the approach.
An extensive ablation study was conducted to isolate the contribution of each component. The improvement in throughput decreases by 12.4% corroborating the advantage of coordination among the agents compared to the sum of the individual contributions without the negotiation protocol (i.e. independent operation of agents). We also evaluated the importance of contribution-proportional allocation by replacing the Shapley-based resource allocation with equal allocation. This resulted in 7.8% reduction of the improvement. Without the phase detection mechanism, the improvement under dynamic workloads was reduced by 18.2%, demonstrating the value of adaptive re-negotiation.

We evaluated stability against workload phase transitions by considering a synthetic workload that switches between OLTP-dominant and OLAP-dominant phases every 30 minutes. MultiTune identified phase transitions in an average of 47 seconds and converged to a new optimal configuration in 3.8 minutes. The degradation during transitions was limited to 8.3% (measured as temporary throughput reduction) compared to 23.7% for OtterTune and 31.2% for the static DBA-Expert configuration. rather than just the framework's ability to predict and adapt to workload shifts reacting to performance degradation proved critical for maintaining stable performance in dynamic environments.

VIII. DISCUSSION AND LIMITATIONS

The results demonstrate that cooperative multi-agent negotiation provides a principled and effective approach to closed-loop database tuning. The key insight is that decomposing the tuning problem into specialised agents each with domain-specific knowledge and optimisation strategies and coordinating through game-theoretic negotiation yields superior performance compared to monolithic optimisation. This decomposition reflects the way expert DBAs think about the tuning problem: not as a single undifferentiated optimization, but as a set of inter-related decisions that need to be coordinated and traded off.

A few limitations have to be discussed. First, the framework used here assumes the fixed set of four agents, which correspond to the four tuning dimensions we identified. Extending the framework to other dimensions like partitioning strategies, parallelism configuration or storage tiering would involve additional agents and a more complex negotiation protocol. The computational complexity of computing the nucleolus grows exponentially in the number of agents, but for large sets of agents this can be reduced by approximation algorithms.

Second, the game-theoretic analysis assumes that agents are cooperative, i.e. they aim at maximizing collective welfare rather than individual utility. This assumption is reasonable in a centrally deployed system, where all agents are under administrative control, but may not hold in multi-tenant or federated database environments where different agents represent different stakeholders that may have conflicting objectives. An important direction for

future work is to extend the framework to non-cooperative or partially cooperative settings.

Third, the evaluation was done on benchmark workloads (TPC-H and TPC-DS) that, while representative of analytical processing, do not capture the full complexity of production workloads. Real world workloads might have more complex temporal patterns, more diversity in queries and stricter latency requirements. Validation on production database systems with appropriate safeguards would strengthen practical claims of the framework.

IX. CONCLUSION

In this paper, we presented a cooperative multi-agent negotiation framework for closed-loop database tuning and index optimization. The framework decomposes the tuning problem into specialized agents and orchestrates them through game-theoretic negotiation protocols, achieving superior performance compared to existing monolithic and single-dimension approaches. On one hand, its theoretical foundation based on cooperative game theory, mechanism design and multi-agent systems provides rigorous guarantees of efficiency, stability and fairness . On the other hand, its practical implementation shows deployability in production environments with minimal overhead .

The larger point of this work is that database tuning, as with many complex systems management problems, is fundamentally a problem of coordination. However sophisticated an optimiser, no single optimiser can capture the full complexity of interactions between tuning dimensions. By embracing this complexity through a multi agent architecture we enable the potential for truly autonomous database systems that adapt, co-ordinate and optimise in real time, freeing the database administrator from routine tuning tasks and allowing them to focus on higher level architectural decisions. Future work will consider extensions to cloud-native and distributed database systems, non-cooperative settings, and integration with learned query optimizers for end-to-end autonomous database management.

BIBLIOGRAPHY

BOOKS

[1] Osborne, Martin J. and Ariel Rubinstein, A Course in Game Theory (MIT Press 1994).

[2] Shoham, Yoav and Kevin Leyton-Brown, Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations (Cambridge University Press 2009).

JOURNAL ARTICLES

[3] Ailamaki, Anastasia and others, 'Database Management Systems: The Next 50 Years' (2022) ACM SIGMOD Record 51(2) 7-14.

[4] Anagnostopoulos, Georgios G. and others, 'Game-Theoretic Approaches to Database Resource Management' (2020) ACM Transactions on Database Systems 45(3) 1-42.

[5] Balkesen, Cagri and others, 'Main-Memory Database Indexing for Modern Hardware' (2013) IEEE Data Engineering Bulletin 36(1) 23-30.

[6] Guha, Sudipto and others, 'Approximation Algorithms for the Nucleolus' (2001) SIAM Journal on Computing 31(2) 438-452.

[7] Li, Guoliang and others, 'A Survey of Machine Learning for Database Systems' (2023) ACM Computing Surveys 55(12) 1-37.

[8] Nash, John F. Jr., 'The Bargaining Problem' (1950) Econometrica 18(2) 155-162.

[9] Schmeidler, David, 'The Nucleolus of a Characteristic Function Game' (1969) SIAM Journal on Applied Mathematics 17(6) 1163-1170.

[10] Shapley, Lloyd S., 'A Value for n-Person Games' in Kuhn and Tucker (eds), Contributions to the Theory of Games, Vol. II (Princeton UP 1953) 307-317.

[11] Zhou, Xuanhe and others, 'A Survey on AI-Driven Database Systems' (2023) ACM Computing Surveys 56(3) 1-38.

CONFERENCE PAPERS

[12] Airoldi, Darrell and others, 'OtterTune: Automatic Database Management System Tuning' (2018) ACM SIGMOD 1-16.

[13] Bruno, Nicolas and Surajit Chaudhuri, 'Automatic Physical Database Tuning: A Relaxation-Based Approach' (2005) ACM SIGMOD 227-238.

[14] Chaudhuri, Surajit and Vivek R. Narasayya, 'An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server' (1997) VLDB 146-155.

[15] Ding, Jialin and others, 'ALEX: An Updatable Adaptive Learned Index' (2020) ACM SIGMOD 969-984.

[16] Kraska, Tim and others, 'The Case for Learned Index Structures' (2018) ACM SIGMOD 489-504.

[17] Marcus, Ryan and others, 'Bao: Making Learned Query Optimisation Practical' (2021) ACM SIGMOD 1275-1288.

[18] Marcus, Ryan and others, 'BtrDB: Bayesian Tuning and Regression for Database Systems' (2020) PVLDB 13(12) 2853-2866.

[19] Mnih, Volodymyr and others, 'Human-Level Control Through Deep Reinforcement Learning' (2015) Nature 518 529-533.

[20] Pavlo, Andy and others, 'Self-Driving Database Management Systems' (2017) CIDR.

[21] Zhang, Tianjun and others, 'Learned Query Optimisation: A Survey' (2022) ACM Computing Surveys 55(6) 1-35.

[22] Zhang, Wei and others, 'IndexTuner: Learning-Based Index Tuning for Dynamic Workloads' (2021) IEEE ICDE 1025-1036.

[23] Zhuang, Zhiqiang and others, 'DQN-Index: Deep Reinforcement Learning for Index Selection' (2022) PVLDB 15(7) 1453-1466.

TECHNICAL REPORTS AND BENCHMARKS

[24] GPyOpt Developers, 'GPyOpt: A Bayesian Optimisation Framework in Python' (2016).

[25] Liu, Xiangyu and others, 'DBTuner: An Automatic Tuning System for PostgreSQL' (2019) IEEE ICDE 1478-1479.

[26] PostgreSQL Global Development Group, 'pg_stat_statements Module Documentation' (2023).

[27] Transaction Processing Performance Council, 'TPC-DS Benchmark Specification v3.2.0' (2022).

[28] Transaction Processing Performance Council, 'TPC-H Benchmark Specification v3.0.1' (2021).