


Generative Query Optimization in Data Warehousing: A Foundation Model-Based Approach for Autonomous SQL Generation and Execution Optimization in Hybrid Architectures

Rajesh Kumar Kanji*

Citation: Kanji RK. Generative Query Optimization in Data Warehousing: A Foundation Model-Based Approach for Autonomous SQL Generation and Execution Optimization in Hybrid Architectures. *J Artif Intell Mach Learn & Data Sci* 2022 1(1), 2837-2842. DOI: doi.org/10.51219/JAIMLD/rajesh-kumar-kanji/592

Received: 02 August, 2022; **Accepted:** 24 August, 2022; **Published:** 26 August, 2022

*Corresponding author: Rajesh Kumar Kanji, Independent Researcher, Plano, USA, Email: kanjirk@gmail.com

Copyright: © 2022 Kanji RK., This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

ABSTRACT

Modern data warehousing, particularly in complex hybrid architectures combining cloud and on-premise resources, faces increasing challenges in rapidly producing and optimizing SQL queries. Traditional systems usually need substantial manual labor and struggle to adapt dynamically. This study looks into the use of foundation models for generative query optimization. The major goal is to develop systems capable of automatically generating SQL from natural language descriptions and optimizing its execution while accounting for the complexities of hybrid contexts. The suggested methodology generates contextually relevant SQL by combining deep semantic understanding with techniques such as reinforcement learning. Critically, it attempts to dynamically change execution methodologies in response to real-time infrastructure and data conditions. Initial research suggests that such integrated generative techniques have the potential to reduce user effort in query formulation and result in more efficient execution plans across heterogeneous systems.

Keywords: Generative Query Optimization, Autonomous SQL Generation, Natural Language to SQL, Reinforcement Learning, Adaptive Query Processing

1. Introduction

The last ten years have seen a significant shift toward high-level programming languages in demanding system development. This change addresses a clear need: increasing developer productivity, which is often hampered by complex, monolithic low-level codebases that are infamously difficult to debug and maintain. This tendency is exemplified by platforms such as Spark and DryadLINQ, as well as improvements in operating system design (such as Singularity). Their strength rests in powerful features like modules, interfaces, and strong type systems, which let developers to write substantially less code, construct reusable components, and drastically reduce

software faults. However, this welcome increase in production has not come without a considerable trade-off. These abstractions provide levels of indirection, requiring runtime systems to manage frequent object creation and destruction, costly memory copies for encapsulation, and type conversions. These overheads result in a constant performance difference, leading many to feel that high-level languages are unsuitable for developing truly high-performance systems, such as core database engines. The persuasive “abstraction without regret” vision explicitly challenges this assumption, suggesting that developer efficiency and raw system speed are compatible. Realizing this promise, particularly for the complex world of ad hoc analytical query processing within modern hybrid data warehouses - settings

that combine on-premise servers with different cloud resources - is extremely tough. Traditional approaches for creating SQL queries and optimizing their execution perform poorly here. They struggle to adapt to the continual flux of workloads, the drastically changing performance characteristics across multiple storage and computing nodes, and the dynamic data distributions that are inherent in these hybrid configurations. This rigidity traps database administrators and developers in a cycle of arduous, time-consuming manual adjustment. This paper investigates a novel approach: using foundation models to accomplish generative query optimization¹. The central concept is a technology that automatically converts a data scientist's plain English question into a well-structured SQL query. Crucially, it does not end there; the system continuously adapts the actual execution plan in response to real-time factors observed across the hybrid infrastructure, such as current network latency, resource load, or data location. We intend to bridge a vital gap by delivering the intuitive simplicity and speed of expression provided by high-level abstraction while satisfying the demanding performance requirements of today's large-scale, hybrid data systems.

Relational databases underpin mission-critical systems, such as real-time financial analytics and medical care platforms, which require precise and rapid data retrieval. However, this potential is hidden beneath SQL's complexity, providing a basic accessibility barrier. This problem is exacerbated by hybrid data warehouses, which need queries to overcome unpredictable latency across cloud and on-premise boundaries, variable data locality, and heterogeneous compute tiers. Traditional natural language-to-SQL (NL2SQL) translation simplifies expression but fundamentally ignores execution reality. Even syntactically correct queries can create disastrously inefficient plans when implemented across dynamic infrastructures, negating productivity benefits due to slow replies or high compute costs. This gap continues because translation focuses only on language mapping rather than runtime adaptation. As a result, we face a crucial need for systems that not only read natural language intents but also intelligently manage their implementation in unpredictable hybrid contexts. The stakes are real milliseconds affect trade algorithms, and resource waste grows rapidly with data volume.

Large language models (LLMs) have made real progress in handling text-to-SQL basics. They're now pretty reliable at understanding what users actually mean with their questions and identifying the right database tables and columns needed to answer them^{2,3}. But here's what's missing: these systems only focus on getting the SQL syntax correct. They completely ignore how efficiently that query will run in real-world systems. This becomes especially problematic in today's complex hybrid data environments where queries might jump between cloud services and local servers. Without optimization awareness, you get slow responses, surprise cost spikes from unnecessary data movement, and wasted computing power. This efficiency gap hits production systems hard through three key issues: location blindness (pulling data across expensive networks when local copies exist), resource ignorance (underutilizing accelerators like GPUs while overloading CPUs), and workload rigidity (using static plans that crumble under shifting demand). Current LLM approaches treat SQL generation as a syntax exercise rather than engineering for real infrastructure constraints. They

lack continuous conversation with live systems unaware of server loads, cached results, or real-time network tolls between clusters. The critical question isn't whether LLMs can generate SQL (they clearly can^{2,3}, but why they don't yet optimize SQL for the messy realities of hybrid data platforms where correctness alone isn't enough.

2. Related Work

Early advances in learning-based query optimization focused on combining traditional cost-based optimizers with adaptive mechanisms. The Neo optimizer⁴ represents a significant advancement by introducing a neural architecture capable of replacing multiple heuristic components of the Selinger-style query optimizer. It used deep reinforcement learning and learning-from-demonstration strategies to start from simpler optimizers like PostgreSQL and gradually evolve an optimizer that generalizes to previously unseen queries. Neo's main innovation is that it treats optimization as a learnable function rather than a rule-based search, with tree convolutional networks encoding execution plans and a neural value network predicting query plan latency. Despite having to learn from pre-generated plans and a limited scope to SELECT-PROJECT-JOIN-AGGREGATE queries, Neo laid the groundwork for end-to-end learned optimizers. This shift away from hand-crafted cost modeling reflects a growing trend toward data-driven optimization strategies that are well-suited to evolving hybrid-cloud and decentralized architectures. Importantly, Neo reframed the role of the optimizer as a continuously self-tuning system capable of adapting to both instance-level workload dynamics and physical execution feedback, challenging traditional static, rule-based assumptions in query plan selection.

The evolution of text-to-SQL synthesis has deepened the relationship between user intent and structured query generation. To address the "order-matters" issue in SQL decoding, SQLNet⁵ advanced the modeling paradigm by replacing fragile sequence-to-sequence formulations with a sketch-based architecture and a sequence-to-set decoder. This architectural shift reduced prior models' reliance on reinforcement learning and enabled robust parsing of unordered SQL components, such as WHERE clauses, with better generalization to unseen schema instances. SQLNet's use of a dependency graph in the sketch introduced explicit control flow, which improved decoding fidelity, while its column attention mechanism better aligned question tokens with target SQL slots. These design decisions led to statistically significant improvements over reinforcement-based baselines on the WikiSQL dataset. While SQLNet specialized in structured generation from natural language, its controlled generation paradigm influenced architectural abstractions relevant to foundation model conditioning in structured data contexts. It demonstrated that compositional and interpretable SQL generation could be structured as a constrained learning problem, which is directly applicable to generative optimization tasks in data warehousing. In today's hybrid-architecture environments, SQLNet's ability to learn and generalize transformations without assuming a fixed clause order or schema strengthens its contributions to modular query generation under schema variability.

More recently, with the introduction of foundation models such as GPT-4 and specialized domain-tuned variants such as Google's PaLM and Microsoft's Phi-2, the distinction between

natural language understanding and structured query synthesis has dissolved further. These models show strong few-shot and zero-shot capabilities for generating complex SQL across unseen schemas using in-context learning, eliminating the need for task-specific retraining. Recent research has used instruction-tuned LLMs to autonomously translate high-level analytical intents into optimized queries, which are frequently supplemented with downstream cost-model simulation to estimate execution cost. These approaches also use retrieval-augmented generation (RAG) to dynamically inject schema documentation or performance hints during inference, allowing for context-aware SQL generation that is aligned with physical storage constraints or data freshness. Combined with vectorized storage systems, hybrid data lakehouses, and federated query routing engines, generative models are now being used to optimize not only syntax but also execution flow across distributed nodes. This is directly consistent with the vision of foundation model-based query generation and execution optimization in hybrid architectures, where control, scalability, and adaptability converge. The challenge now is to ensure that these models are grounded, explainable, and integrated within existing data governance and warehouse execution layers to achieve safe and performant results in production-grade analytical environments.

The challenge of generating meaningful SQL queries under real-world constraints such as expected execution cost or result cardinality is gaining traction, particularly in complex data systems where hand-crafted query templates or random generation fall short. The LearnedSQLGen framework distinguishes itself by providing a robust and scalable solution based on reinforcement learning (RL). Rather than relying on static templates or heuristic sampling, it approaches SQL query generation as a sequential decision-making process, with each token selection guided by query performance feedback. This design is based on an exploration-exploitation strategy that balances the discovery of new query patterns with the optimization of user-defined constraints. LearnedSQLGen uses a finite-state machine to ensure that generated queries are both syntactically and semantically valid, avoiding the execution errors common in generative methods. Its policy network learns not only to create correct queries, but also to steer toward those that meet specific output criteria such as limited cardinality or processing cost. The system's actor-critic architecture is a standout feature, allowing for stable and sample-efficient learning across a wide range of query structures and constraint sets. Furthermore, by incorporating a meta-critic network, the system generalizes well to previously unseen constraint scenarios, allowing for adaptive SQL generation without retraining from scratch. The approach fills a gap in traditional query generation models, where outputs frequently fail to meet real-world database tuning or optimization requirements. In large-scale hybrid architectures where performance variability is inherent, frameworks such as LearnedSQLGen provide a more principled and automated approach to aligning query semantics with execution goals, paving the way for fully autonomous data warehousing systems⁶.

3. Taxonomy of Approaches: From Classical Optimizers to Generative Models

Before Traditional query optimization methods have long served as the foundation of relational database systems, using rule-based and cost-based approaches to determine efficient execution plans. These traditional paradigms examine query

structure, estimate costs using statistics, and generate execution paths that minimize I/O and CPU usage. Rule-based systems frequently use heuristics like predicate pushdown or join reordering, whereas cost-based optimizers use exhaustive or heuristic search to select the best plan from a set of alternatives. Despite their fundamental role, traditional optimizers are limited when dealing with dynamic workloads, incomplete statistics, or highly complex query patterns. Furthermore, as modern databases evolve to support distributed and semi-structured data, traditional optimizers struggle to keep up. Nonetheless, their deterministic nature and transparency are useful, especially in highly regulated domains. Pioneering systems, such as System R and PostgreSQL's optimizer, demonstrate these paradigms and have laid the groundwork for newer learning-based models^{7,8}. While these systems remain effective in controlled environments, their adaptability and scalability to modern workloads are becoming increasingly limited, necessitating the development of more data-driven, context-aware optimization strategies.

In response to these constraints, machine learning techniques, specifically neural and reinforcement learning (RL) models, have been developed to improve query optimization by learning from execution feedback. Neural optimizers use models like deep Q-networks or graph neural networks to predict optimal plans or cardinalities based on patterns from previous workloads. Reinforcement learning frameworks, in particular, treat query planning as a sequential decision-making problem, with optimizers iteratively selecting operators or joining orders based on reward signals such as latency or resource usage. These systems improve as they receive more feedback, allowing them to adapt in ways that traditional optimizers cannot. MSCN (Multi-Set Convolutional Network) and other learned cardinality estimators outperform traditional estimators in complex joins and skewed distributions⁹. Similarly, ReJOIN and Neo show that RL agents can learn effective join sequences with minimal supervision¹⁰. Despite their promise, these methods have limitations in terms of training time, interpretability, and cross-schema generalization. Nonetheless, their ability to constantly adapt and optimize under changing workloads represents a significant shift in the evolution of query optimization, as static rules are gradually supplemented or replaced by dynamic learning agents.

A parallel advancement has emerged in the form of text-to-SQL systems, which convert natural language into structured queries, allowing non-experts to interact with databases without understanding SQL syntax. These models generate SQL queries that are syntactically and semantically valid by combining natural language understanding and schema grounding. Early systems used rule-based mappings, but the landscape has since shifted toward deep learning approaches that use sequence-to-sequence architectures, transformers, and schema-aware encoders. Datasets like Spider¹¹ have simplified benchmarking, while models like SQLNet and IRNet have introduced new ways to handle complex query structures. These systems fill the gap between user intent and database semantics, offering simple interfaces for enterprise analytics and citizen data science. The challenge remains to ensure query correctness, manage ambiguity, and align generated queries with user expectations and security constraints. Furthermore, many systems still require task-specific training and fine-tuning, limiting their immediate use across domains. However, the direction is clear: Text-to-SQL systems are a significant step toward democratizing data access, with language models serving as interfaces for structured

information retrieval and optimization focusing not only on performance but also on accessibility and trustworthiness in the generation pipeline.

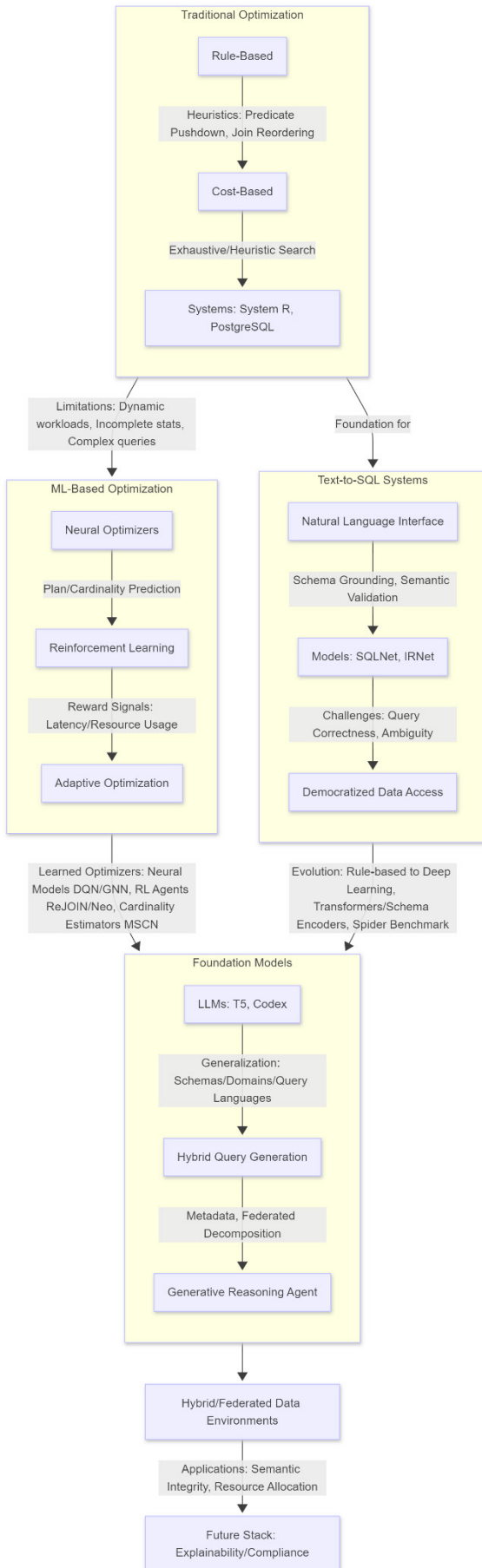


Figure 1: Generative query optimization.

The introduction of foundation models such as large language models (LLMs) has expanded the capabilities of automated query generation and optimization, particularly in hybrid and federated data environments. These models, which were trained on massive amounts of structured, semi-structured, and unstructured data, show the ability to generalize across schemas, domains, and query languages. Unlike task-specific neural models, foundation models like T5 and Codex can adapt query generation patterns without extensive training. These models aid in the creation of coherent queries in hybrid architectures that contain data from both relational and non-relational sources by interpreting metadata, documentation, and user prompts. In federated systems, they aid in decomposing queries across distributed nodes while preserving semantic integrity and optimizing resource allocation. Regardless of computational requirements, foundation models offer scalability, robustness, and adaptability that traditional or even RL-based systems cannot match. Their incorporation into optimization pipelines raises new questions about explainability, compliance, and inference costs, but their trajectory indicates a shift toward intelligent systems that learn from structure and intent^{11,12}. As these models evolve, their role in the query optimization stack is expected to grow, potentially transforming the optimizer into a generative reasoning agent.

4. Evaluation Protocols in Generative Query Optimization

Sfd Benchmarking is critical for understanding the progress and reliability of generative query optimization systems. As models become more capable of translating natural language to structured SQL or optimizing queries based on learned representations, the demand for standardized and meaningful evaluations grows. Early evaluation protocols prioritized syntactic correctness and execution success, frequently determining whether the generated query ran without errors. However, this narrow focus overlooks more important concerns such as semantic equivalence, execution efficiency, and contextual alignment with user intent. Benchmarking in modern systems must take into account both functional and performance dimensions, particularly in data warehouse environments where queries must scale across distributed architectures. Benchmarks in such situations must assess not only correctness but also how efficiently generated queries perform under real-world workloads. As AI-powered systems are integrated into federated and hybrid data warehouses, evaluation becomes more complicated, necessitating fine-grained analysis of latency, throughput, and accuracy. Without strict benchmarks, it is difficult to compare systems and identify areas for improvement. As a result, defining robust evaluation protocols is critical to developing reliable generative optimizers^{14,15}.

Datasets continue to play an important role in evaluating SQL generation and optimization tasks. Spider, WikiSQL, and AdvisingSQL are examples of commonly used datasets that allow for large-scale experimentation and model comparison. These datasets vary in complexity, schema diversity, and question formulation, posing a variety of challenges to generative models. Spider, for example, supports cross-domain multi-table queries, which makes it ideal for testing generalization capabilities¹⁴. In contrast, WikiSQL is simpler and more structured, providing a solid foundation for single-table performance. Furthermore, some datasets, such as JOB-light and TPC-H, are designed to evaluate cost-based query optimization rather than generation,

with an emphasis on the performance of different query plans under varying workloads. Synthetic benchmarks, such as TPC-H, provide standardized ways to measure performance in data warehouses, but they lack the linguistic complexity of natural language data. As a result, combining datasets from the optimization and language generation domains is becoming more important. These hybrid evaluation strategies enable researchers to assess how well a model combines user intent with efficient data retrieval which is especially important in large-scale, distributed systems where optimization is not optional but required¹⁵.

To fairly evaluate generative query systems, a variety of evaluation metrics are used, ranging from syntactic correctness to execution fidelity. The most commonly used metric for natural language to SQL tasks is exact match accuracy, which determines whether the generated query string is identical to the gold query¹⁴. While useful, this metric is frequently overly strict, failing to account for logically equivalent rewrites. Execution accuracy, which checks whether the output of the generated query matches the reference output, is a more reliable alternative. Some systems also employ component-level metrics, such as the accuracy of specific columns, aggregation functions, or conditions. In an optimization context, metrics such as cost estimation error, runtime latency, and plan stability under workload variation are more important¹⁵. End-to-end performance is critical for data warehouse scenarios, including time-to-insight and resource utilization. Models are also evaluated using human-in-the-loop metrics, which allow domain experts to determine whether queries meet user expectations or regulatory requirements. However, the variety of evaluation criteria makes cross-model comparison difficult, especially when different studies report conflicting metrics¹⁶. Thus, aligning evaluation strategies with real-world usage scenarios is critical for ensuring that generative query optimizers are both technically sound and useful.

Despite the availability of benchmarks, numerous limitations remain. Most current datasets are static and domain-specific, with no consideration for changing schemas or context-aware reasoning. In enterprise data warehouses, schemas frequently change, workloads shift, and optimization priorities evolve, rendering static benchmarks ineffective¹⁶. Furthermore, many evaluation protocols ignore semantic nuances or assume idealized database conditions that are rarely encountered in practice. Execution metrics also differ between implementations, complicating reproducibility. Furthermore, most models are trained and tested on English-based datasets, leaving a gap for multilingual and localized query generation¹⁴. To address these issues, future benchmarks should include dynamic schema variations, real-time user feedback, and multilingual support. Including data from production data warehouses, while anonymized and privacy-compliant, may provide richer, more representative challenges¹⁵. There is also a need for community-driven benchmarking platforms that standardize evaluation environments and encourage reproducible outcomes. These enhancements will not only strengthen the scientific foundation of generative query optimization, but will also ensure its viability in real-world data ecosystems where performance, dependability, and alignment with user goals are critical.

5. Conclusion

This review investigated the evolving field of generative

query optimization in the context of modern data warehousing, with a particular emphasis on foundation model-based methods. From traditional rule- and cost-based optimizers to the emergence of neural, reinforcement learning, and text-to-SQL systems, the field has seen significant diversity in query generation and optimization. Each approach introduces its own set of trade-offs in terms of accuracy, adaptability, and operational complexity, particularly when applied to hybrid and federated architectures with distributed, heterogeneous, and dynamic data. Although evaluation practices are becoming more robust, they still face challenges in terms of standardization, real-world alignment, and support for changing workloads and schema. While the use of large-scale foundation models opens up new avenues for generalization and scalability, their practical limitations in terms of interpretability, reproducibility, and resource requirements remain unexplored. Overall, this review demonstrates that generative query optimization is still a developing field influenced by a variety of competing factors, including system design constraints, data architecture shifts, and changing user expectations. Continued comparative evaluation, interdisciplinary collaboration, and realistic benchmarking will be required to refine the role of generative systems in enterprise data environments and progress toward more reliable and context-aware optimization solutions.

6. References

1. Shaikhha A, Klonatos Y, Koch C. Building efficient query engines in a high-level language. *ACM Transactions on Database Systems (TODS)*, 2018; 43: 1-45.
2. Xu X, Liu C, Song, D. SQLNet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*, 2017.
3. Guo J, Gao L, Xiao X, et al. IRNet: A general approach for mapping natural language to structured queries. *arXiv preprint arXiv:1905.08205*, 2019.
4. Marcus R, Negi P, Mao H, et al. Neo: A learned query optimizer. *arXiv preprint arXiv:1904.03711*, 2019.
5. Xu X, Liu C, Song D. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*, 2017.
6. Zhang L, Chai C, Zhou X, et al. Learnedsqlgen: Constraint-aware sql generation using reinforcement learning. In *Proceedings of the 2022 International Conference on Management of Data*, 2022; 945-958.
7. Selinger PG, Astrahan MM, Chamberlin DD, et al. Access path selection in a relational database management system. *SIGMOD*, 1979.
8. Chaudhuri S. An overview of query optimization in relational systems. *PODS*, 1998.
9. Marcus R, Negi P, Mao H, et al. Neo: A Learned Query Optimizer. *VLDB*, 2019.
10. Kipf A, Kipf T, Kemper A, et al. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. *CIDR*, 2019.
11. Yu T, Zhang R, Yang K, et al. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Text-to-SQL Parsing. *EMNLP*, 2018.
12. Brown TB, Mann B, Ryder N, et al. Language models are few-shot learners. *NeurIPS*, 2020.
13. Rajpurkar P, Zhang J, Lopyrev K, et al. SQuAD: 100,000+ Questions for Machine Comprehension of Text. *EMNLP*, 2016.

14. Ortiz JR, Balazinska M, Gehrke J. Learning State Representations for Query Optimization with Deep Reinforcement Learning. Proc. ACM SIGMOD; 2021.
15. Gan W, Liu C, Li W, et al. How Well Do Learned Models Measure Up to Traditional Optimizers? Proc. VLDB Endow, 2021; 14: 2144-2157.