软件学报 ISSN 1000-9825, CODEN RUXUEW
[doi: 10.13328/j.cnki.jos.007659] [CSTR: 32375.14.jos.007659]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

QueryGuide: 基于生成式模型的细粒度查询提示推荐方法*

杨少聪[1,2], 王 宁[1,2]

[1](北京交通大学 计算机科学与技术学院, 北京 100044)

[2](交通数据挖掘与具身智能北京市重点实验室, 北京 100044)

通信作者: 王宁, E-mail: nwang@bjtu.edu.cn

摘 要: 慢查询是指执行时间超过数据库系统所设定的阈值, 从而影响整体性能的查询. 由于开发人员经验不足及查询接口模板缺乏灵活性, 慢查询在应用程序中频繁出现, 成为数据库故障的主要原因. 为了提速慢查询, 数据库管理员通常会进行人工修复, 但是在数据量较大的情况下效率较低, 因此针对慢查询的自动重写方法的研究显得尤为重要. 查询提示是用于显式控制数据库优化器决策的调优指令, 可以通过影响访问路径等策略来改善查询性能. 查询提示推荐作为数据库查询重写领域的重要研究问题, 已经得到广泛关注. 然而, 目前基于机器学习的提示推荐方法仍面临粒度较粗、依赖人工指定候选集等问题. 针对这些问题, 提出基于生成式模型的细粒度提示推荐框架 QueryGuide, 通过探索不同细粒度提示的最优组合方式, 来提升重写语句的物理计划执行效率. 具体而言, 该框架一方面基于生成式语言模型探寻推荐策略, 在数据库环境中探索提示对慢查询性能的提升, 并将其作为奖励用于模型微调, 从而生成表级别的细粒度提示; 另一方面提出使用生成式模型推荐的细粒度提示作为探索方向, 并通过扩展得到提示组合. 将组合的探索结果用于模型微调, 可以使模型学习不同提示组合对慢查询重写性能的影响, 从而获得最佳提示组合. 在多个数据集上的实验结果表明, 相较于其他基线算法, QueryGuide 方法表现出更好的重写性能, 显著改善了查询时间.

关键词: 查询重写; 慢查询优化; 查询提示; 生成式语言模型

中图法分类号: TP311

中文引用格式: 杨少聪, 王宁. QueryGuide: 基于生成式模型的细粒度查询提示推荐方法. 软件学报. http://www.jos.org.cn/1000-9825/7659.htm

英文引用格式: Yang SC, Wang N. QueryGuide: Fine-grained Query Hint Recommendation Method via Generative Model. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7659.htm

QueryGuide: Fine-grained Query Hint Recommendation Method via Generative Model

YANG Shao-Cong[1,2], WANG Ning[1,2]

[1](School of Computer Science and Technology, Beijing Jiaotong University, Beijing 100044, China)

[2](Beijing Key Laboratory of Traffic Data Mining and Embodied Intelligence, Beijing 100044, China)

Abstract: A slow query refers to a query whose execution time exceeds the threshold defined by the database system, thus degrading overall performance. Due to limited developer experience and inflexible query interface templates, slow queries frequently occur in applications and are a major cause of database failures. To accelerate slow queries, database administrators often perform manual tuning. However, manual tuning is inefficient for large volumes of data, making research on automatic query rewriting methods for slow queries particularly important. Query hints are tuning directives that explicitly influence optimizer decisions and can improve query performance by affecting strategies such as access paths. As an important research problem in the field of database query rewriting, query hint recommendation has received widespread attention. However, existing machine learning-based hint recommendation methods still face issues such as coarse granularity and reliance on manually specified candidate sets. To address these issues, this study proposes

* 基金项目: 国家自然科学基金 (62372034)
收稿时间: 2025-07-05; 修改时间: 2025-11-10, 2026-01-14; 采用时间: 2026-03-03; jos 在线出版时间: 2026-06-10

QueryGuide, a fine-grained hint recommendation framework based on a generative model. By exploring the optimal combinations of different fine-grained hints, the execution efficiency of physical plans for rewritten queries is improved. Specifically, on the one hand, the framework explores recommendation strategies based on generative language models, investigates the performance improvement of hints on slow queries in database environments, and uses it as rewards for model fine-tuning, thus generating fine-grained, table-level hints. On the other hand, the fine-grained hints recommended by the generative model are used as exploration directions, and hint combinations are obtained through expansion. The exploration results of these combinations are used for model fine-tuning, enabling the model to learn the impact of different hint combinations on slow query rewriting performance and to identify the optimal hint combination. Experimental results on multiple datasets demonstrate that, compared with other baseline algorithms, the proposed method exhibits better rewriting performance and significantly reduces query execution time.

Key words: query rewriting; slow query optimization; query hint; generative language model

数据库的查询执行时间直接影响应用系统的响应速度, 因此查询优化一直是数据库领域的重要研究问题[1−5]. 传统的数据库优化器使用重写规则对查询计划进行等价转换, 得到一个执行效率更高的查询计划[6−9]. 这种基于启发式规则的优化往往能够在时间和计算资源有限的情况下完成查询任务, 因此在不同的数据库环境下广泛使用.

互联网时代, 应用程序产生的数据量日益庞大. 据统计, 在 2023 年, Twitter 平台每天会生成大约 5 亿条推文, Facebook 的数据处理量超过 1.7 亿 GB. 在处理大数据量时, 基于启发式规则的传统数据库优化器由于缺乏足够的可扩展性, 无法考虑到多级优化和跨多个表的联合优化, 会导致查询时间过长而形成慢查询, 对应用平台造成不利影响. Mobify 研究发现, 查询时间每增加 100 ms, 用户留存率就会减少 1.55%, 进而年收入下降近 530 000 美元. 为了保证用户的满意度, 在高并发、高负载的环境中如何保证查询的执行效率是一个亟待解决的问题.

AI 大模型时代, 机器学习算法由于其强大的学习能力和可扩展性, 为数据库的查询优化提供了新思路和新方法. 在人工智能赋能的数据库查询优化领域, 目前在基数估计、代价估计、索引选择和旋钮调优的研究方向都有新的进展[2,10]. 然而, 这些研究方法都需要在数据库管理系统 (database management system, DBMS) 中集成机器学习模型, 需要较高的技术要求和复杂的工程实现, 因此在工业界较难落地[11]. 为了提高算法的实用性, 研究人员在查询重写领域进行了深入研究. 查询重写是数据库查询优化的关键步骤, 旨在将原始的查询语句转化为一个与之语义等价且执行效率更高的查询语句[6,12]. 它将查询优化器看作黑盒, 采用外部的优化方式来提高查询执行效率, 可操作性好且易部署.

查询提示是对慢查询进行物理层面优化的重写方法. 通过在查询语句中直接指定计划节点的物理执行方式, 来引导数据库优化器生成性能最优的执行计划, 可以解决由于传统优化器难以准确估计查询代价而导致的查询缓慢问题[11,13]. 为了提高数据库管理员 (database administrator, DBA) 为慢查询添加查询提示的效率, 有必要提供自动的提示推荐方法, 自动感知欠优化的查询节点, 并推荐提示来将节点引导至最优操作方式. 现有的提示推荐方法按其实现方式可分为基于强化学习的算法和基于启发式的算法. 基于强化学习的方法通过模拟数据库运行环境, 训练智能体在多个查询提示集中选择最优的提示集. 基于启发式的方法则在多个查询提示中探索最优的提示组合方式. 现有方法能够在一定程度上为慢查询带来性能提升, 但仍存在两方面的问题.

1) 探索和推荐的提示粒度比较粗, 导致性能提升有限. 现有方法的探索空间均局限于布尔类型提示 (如是否启用并行执行), 使用较粗的粒度来规划查询计划的操作方式, 会限制其在复杂场景中的应用效果. 如果为查询计划的多个节点分配不同操作方式往往会有更大的性能提升空间, 因此如何探索表级别的细粒度提示 (如对特定表使用索引扫描) 是一个挑战.

例如, 在 JOB 基准测试中, 查询 17b 的执行时间约为 18 s, 其性能瓶颈出现在多表连接操作 $k\_mk \bowtie ci \bowtie \sigma\_\{n.\text{name LIKE 'Z\%'}\}(n)$ 上. 如图 1 所示, 查询优化器采用嵌套循环策略, 依次连接表 *ci* 和表 *n*. 表 *n* 的数据量达百万级别, 但未进行谓词提前过滤, 导致连接时大量不必要的磁盘 I/O 操作, 严重影响了查询效率. 为提高访问效率, 粗粒度提示引入哈希连接和顺序扫描操作. 该策略将表 *n* 过滤后的记录加载至内存哈希表, 并与表 *ci* 进行连接. 该优化使查询计划的时间缩短至 8 s. 然而, 粗粒度提示优化方法存在一定的局限性: 强制表 *ci* 采用顺序扫描, 无法有效利用该表上的索引. 当表 *ci* 的数据量达千万级别的情况下, 这种访问方式仍存在性能瓶颈. 为解决该问题可引入细粒度提示, 通过单独对表 *n* 设置顺序扫描操作, 会使查询优化器将其相关的连接进一步优化为哈希连

接操作. 这种方式同时实现了对表 *n* 的谓词提前过滤和对表 *ci* 的索引利用, 执行时间进一步降低至 3 s, 对比粗粒度方案, 性能提升了 62.5%. 可见, 与粗粒度提示相比, 细粒度提示能更灵活地调整查询优化策略, 因此具有更大的性能提升潜力.

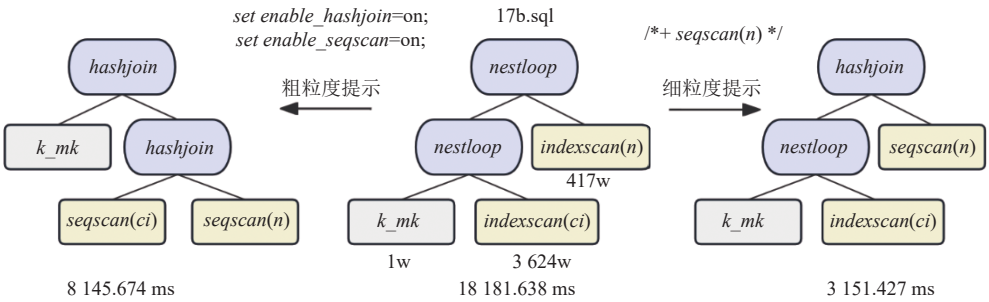

图 1　不同提示粒度在慢查询优化方式上的对比

2) 候选提示集依赖经验丰富的专家生成, 人工成本较高. 现有方法均需专家提前给定候选集, 然后使用算法在其中择优推荐, 无疑这增加了使用门槛和维护成本, 因此如何简化人为干预、自动生成查询提示是一个挑战.

针对上述挑战, 本文提出一种基于生成式模型的细粒度查询提示推荐方法 QueryGuide. 该方法一方面在对基于预训练的生成式模型微调的基础上, 探索细粒度查询提示的生成策略, 进而自动推荐出可规划具体节点的查询提示; 另一方面, 生成式模型在数据库环境中对提示组合进行探索和微调, 能够学习到不同组合对慢查询的性能提升水平, 从而生成最佳细粒度提示组合, 在简化建模成本的同时提升查询性能.

综上所述, 本文的贡献如下.

(1) 提出一种细粒度的查询提示推荐框架. 该框架利用生成式模型的学习和泛化能力, 实现了细粒度提示的自动组合与优化, 能够对慢查询计划的欠优化节点实施精细化调优.

(2) 提出基于生成式语言模型探索细粒度提示的方法. 基于生成式模型以及微调, 能够实现对细粒度查询提示的探索. 该模型无需提前指定候选集, 减少了人力成本.

(3) 提出受强化学习启发的迭代式提示组合探索方法. 通过在 DBMS 中探索提示组合的性能提升效果, 推荐模型可以学习到不同组合的性能提升能力, 显著提高细粒度提示带来的查询性能提升.

本文第 1 节介绍查询提示推荐研究的基础知识以及相关工作. 第 2 节对细粒度查询提示推荐问题进行形式化定义, 并建立马尔可夫过程模型. 第 3 节介绍算法的总体框架, 并简要描述推荐模型的微调和探索过程. 第 4 节介绍细粒度提示推荐策略, 通过使用带有奖励的样本数据对生成式模型微调, 来提高推荐模型生成提示的作用效果. 第 5 节介绍将生成式模型与强化学习思想相结合的提示组合探索策略. 通过将推荐模型生成的提示作为探索方向进行提示组合扩展, 并在数据库环境中探索收益, 来使推荐模型学习到不同提示组合的性能. 第 6 节在多个数据集上进行实验, 通过实验结果来定量分析模型的有效性. 第 7 节总结全文.

1　相关工作

作为物理层面的查询优化方法, 查询提示技术通过将提示作为前缀添加到查询语句中, 可引导优化器生成最优的物理执行计划.

1.1　查询提示的作用方式

查询提示是 DBMS 为 DBA 提供的优化接口, 是对 SQL 语句调优的重要工具. 数据库优化器依赖统计信息和成本模型选择执行计划, 这种启发式方法可能会生成次优的查询计划. 当传统优化器无法为查询生成高效的执行计划时, 通过为查询语句添加查询提示前缀, 用提示覆盖优化器的默认选择, 即对查询计划的执行方式进行人为干预, 可实现查询计划的进一步调优. 依据作用方式, 可将查询提示分类为操作类提示和策略类提示.

操作类提示用于引导优化器设定计划节点的物理操作. 根据作用范围, 可以将操作类提示进一步分为两类. 第 1 类是布尔类型的粗粒度提示, 对查询计划进行全局优化. 该类提示通过启用和禁用物理操作符, 以较粗的粒度引导优化器去规划物理计划. 如设置 *set enable_hashjoin* = on 表示允许查询计划中采用哈希连接操作, 设置 *set enable_seqscan* = off 表示禁止查询计划中采用顺序扫描操作. 该类提示可视为 DBMS 的旋钮配置, 可用于对工作负载中的多条查询进行整体调优. 第 2 类是表级别的细粒度提示, 对查询计划进行局部优化. 该类提示通过为具体的计划节点指定物理操作方式, 来精准引导优化器规划物理执行计划. 如设置 *hashjoin*(*a b*) 表示表 *a* 和表 *b* 使用哈希连接操作, 设置 *seqscan*(*a*) 表示对表 *a* 使用顺序扫描操作, 设置 *Leading*(((*c a*) *b*)) 表示 *a*、*b*、*c* 这 3 张表之间采用的连接顺序为 *c* 连接 *a* 再连接 *b*. 操作类提示直接作为前缀添加到查询语句前, 可用于对查询语句的重写优化.

策略类提示用于引导优化器对查询计划设定策略, 可分为 3 类. 第 1 类是并行查询提示, 能够显式设置查询语句的并行执行策略. 如设置 PARALLEL HINT 用于指定查询的并行度, 覆盖初始化参数 PARALLEL_DEGREE_POLICY, 且当查询涉及排序或分组操作时, 实际使用的线程数可达指定值的两倍; 设置 NO_USE_PX 表示禁止查询使用 PX 模式, 以确保其在单线程模式下执行. 第 2 类是查询转换提示, 通过转换查询结构来提高执行效率. 如设置 NO_REWRITE 禁用查询重写; 设置 NO_EXPAND 会阻止优化器将 OR 条件拆分为多个查询执行路径; 设置 PLACE_GROUP_BY 会允许优化器进行 GROUP BY 位置转换, 即优化器可能调整 GROUP BY 的计算位置, 以减少数据传输和计算成本. 第 3 类是查询策略提示, 能够设置优化器在即时编译 (just-in-time, JIT)、物化、聚合等方面的策略. 如设置 USE_LATE_MATERIALIZATION 能够推迟物化操作, 仅在必要时进行数据计算; 设置 USE_JIT 能够启用 JIT 编译模式, 提高查询执行性能.

1.2 查询提示的推荐模型

现有的提示推荐模型可以分为两类. 第 1 类是基于强化学习的推荐模型, 在多个查询提示候选集中选择最优提示集; 第 2 类是基于启发式的推荐方法, 在多个查询提示中探索最优组合.

(1) 基于强化学习的推荐模型

为了辅助数据库优化器生成执行成本更低的物理计划, Bao 模型[11]最先提出为慢查询自动探索最优的查询提示集. 给定提示集的候选集合, 该模型将查找最优提示集的问题建模为多臂老虎机问题, 并将每个提示集看作是老虎机的一只手臂. 另外, 该模型引入树卷积神经网络来对添加提示的查询语句进行成本预测, 并以此为依据选择最优提示. 这种方法能够为慢查询探索最优提示集, 得到最优的物理查询计划. 然而, 该模型需要在整个提示集空间内进行搜索, 因此耗费的时间较长. 为了提高搜索效率, Maliva 模型[14]增加了时间限制, 通过将探索状态 $s=(E,C_i,T_i)$ 与深度 Q 学习算法相结合来对探索序列做出决策, 以此实现搜索时间和搜索质量的平衡. 上述方法都是对粗粒度布尔类型的查询提示进行探索, 没有考虑查询计划中表之间的连接顺序. 为了解决这个问题, HyperQO 模型[15]提出对 Leading 类型提示进行探索, 使用蒙特卡洛树搜索与学习型估计器相结合的方式探索最优的表连接顺序, 减少由于不当的表连接顺序所增加的查询成本. 然而, HyperQO 模型采用的成本估计器存在不确定性, 因此探索出的提示集性能也存在不确定性. 为了提高重写性能的稳定性, COOOL 模型[16]提出引入学习型排序算法, 使用查询计划之间的相对排名来代替绝对成本. 该模型将树卷积网络和多层感知机相结合作为排序模型, 用于对提示集性能的排名预测. 由于相对的性能排序比绝对的性能预测更加具有可靠性, 因此添加提示集的重写语句会具有更稳定的性能.

(2) 基于启发式的推荐模型

基于强化学习的推荐方法需要数据库专家提前指定查询提示组合候选集, 为减少人力成本, AutoSteer[17]使用模型来对多个查询提示的最优组合进行自动探索. 该模型运用贪心策略与剪枝策略相结合的方式对最优提示组合进行剪枝、合并和探索, 并使用树卷积网络作为提示组合的成本估计器. AutoSteer 无需专家指定候选提示集, 尽管实现的门槛较低, 但由于采用贪心策略进行剪枝, 很可能陷入局部最优. 为了提高探索到最优提示组合的概率, FASTgres 模型[18]提出使用上下文感知的分类策略. 该模型首先将慢查询根据连接表和连接属性进行类别划分, 然

后为每个类别分别建立基于 XGBoost 的最优组合预测器, 来实现对慢查询最优提示组合的精准探索. FASTgres 对查询进行细粒度划分以建立模型, 因此生成的提示组合具有更少的性能退化现象.

虽然上述两类方法都对查询提示组合进行了有效探索, 但是这些方法都局限于推荐布尔类型的提示, 无法对具体的计划节点进行最优规划, 并且需要人为指定候选提示集, 这在一定程度上限制了重写语句的优化效果.为了减少人为干预同时提升重写语句性能, 本文提出了一种基于生成式模型的细粒度查询提示推荐方法 QueryGuide. 该方法通过对基于预训练的生成式模型的微调, 能够针对具体节点推荐细粒度的查询提示. 此外, 通过对提示组合进行探索和微调, 能够学习到不同提示组合对慢查询的性能提升水平, 因此 QueryGuide 能够在简化建模成本的同时提升查询性能.

2 问题模型和相关定义

定义 1. 细粒度查询提示. 细粒度查询提示是指直接作用于查询优化器的物理执行层, 并对表级别的物理操作进行精准控制的提示类型. 细粒度查询提示的作用方式包括指定单表的访问策略以及多表的连接方式. 给定一个查询 q 及其涉及的表集合 T, 可以运用的细粒度查询提示如下.

(1) 单表访问策略

$$S = \{seqscan(t), indexscan(t), indexonlyscan(t) \mid t \in T\}.$$

(2) 多表连接方式

$$J = \{hashjoin(t_i, t_j), mergejoin(t_i, t_j), nestloop(t_i, t_j) \mid t_i, t_j \in T\}.$$

与粗粒度查询提示相比, 细粒度的提示能够对执行计划的关键步骤实施表级别的明确约束, 提供更高的优化可控性.

定义 2. 细粒度查询提示组合. 给定一个查询语句 q, q 涉及的表集合为 T, 如果 T 上可以运用单表访问策略集合 S 和多表连接方式集合 J, 则该查询可以运用细粒度查询提示集 FH:

$$FH = S \cup J.$$

在不冲突的情况下, FH 中的查询提示可以组合使用. q 可以运用的细粒度查询提示组合集 FS 为:

$$FS = \{fs \mid fs \subseteq FH\}.$$

定义 3. 细粒度查询提示最优组合. 给定一个查询语句 q 以及 q 上可以运用的细粒度查询提示组合集 FS, 在查询 q 中添加某个提示组合 $fs_i \in FS$ 可以得到重写语句 q^{fs_i}, 该提示组合将查询 q 的计划由 $plan_q$ 调整为 $plan_{q^{fs_i}}$, 调整后的执行成本为 $cost(plan_{q^{fs_i}})$. 对于查询 q 而言, fs_* 为细粒度查询提示最优组合, 当且仅当添加该提示组合后得到调整后的执行计划具有最低成本:

$$m = \arg\min_i cost(plan_{q^{fs_i}}),$$

$$fs_* = fs_m.$$

定义 4. 细粒度查询提示最优组合探索问题. 给定一个查询语句 q, 其执行成本为 $cost(q)$, 可以运用的细粒度查询提示组合集为 FS. 细粒度查询提示最优组合探索问题就是在 FS 中探索出最优组合 fs_*, 使得重写语句 q^{fs_*} 具有最低的执行代价.

运用强化学习算法为查询探索最优提示组合, 会执行下列过程.

(1) 探索模型为当前查询 q 推荐提示 fs.

(2) 将提示 fs 运用到查询 q 上得到重写语句 q^{fs}, 并通过与 DBMS 交互得到提示带来的性能收益.

(3) 探索模型根据性能收益对提示组合的推荐策略进行更新.

我们将该过程形式化为强化学习的马尔可夫决策过程模型:

$$MDP = (Q, FS, C).$$

状态空间 Q 是指在提示组合探索过程中查询 q 可以转换得到的重写语句, 可以表示为:

$$Q=\{q^{fs}\mid fs\in FS\}.$$

动作空间 FS 是 q 可应用的细粒度提示组合集合. 动作 $fs\in FS$ 是一个将状态 q 映射到新的状态 q^{fs} 的函数, 即:

$$fs(q)\colon q\to q^{fs}.$$

奖励函数 $C(q,fs)$ 表示在应用提示组合 $fs\in FS$ 后查询 q 的执行成本减少量:

$$C(q,fs)=c(q)-c(q^{fs}).$$

DBMS 为强化学习中的环境, 当查询 q 及其重写语句 q^{fs} 在 DBMS 中执行, 获得的执行成本分别是 $c(q)$ 和 $c(q^{fs})$.

探索模型 π 为强化学习中的智能体. 对于给定状态 $q_i\in Q$, 所有提示组合 $fs\in FS$ 都被探索模型 π 作为可选择的动作, 即:

$$\pi\colon Q\to FS.$$

细粒度查询提示最优组合探索问题的目标是找到一个探索模型 π^*, 能够为慢查询 q 推荐合适的提示动作 fs_*, 以获得性能最优的重写语句 q^{fs_*}:

$$\pi^*=\arg\max_{\pi}C(q,\pi(q)).$$

3 总体框架

为了提高查询提示带来的性能提升, 我们提出了一个基于生成式模型的细粒度提示推荐框架 QueryGuide, 通过探索最优的细粒度提示组合, 将慢查询的物理执行计划引导至最佳路径. QueryGuide 框架利用谷歌提出的生成式语言模型 T5 (text-to-text transfer Transformer)[19]来对查询语句进行编码, 并预测和推荐合适的查询提示组合. 如图 2 所示, QueryGuide 的总体框架包含两部分: 查询提示生成模块、提示组合探索模块.

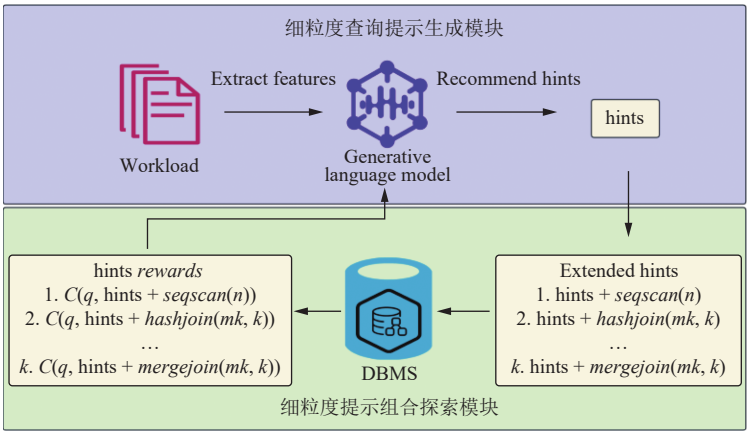

图 2 基于生成式模型的细粒度查询提示推荐框架 QueryGuide

查询提示生成模块对慢查询进行特征分析和提取, 为其推荐查询提示. 该模块充分利用生成式模型的预训练知识, 对查询提示组合的优化策略进行高效学习.

提示组合探索模块通过与 DBMS 交互, 进一步探索和优化多个查询提示的组合效果. 该模块首先利用查询提示生成模块生成细粒度提示, 然后将该提示作为探索方向进行提示组合的扩展式搜索. 通过探索更多可能的提示优化策略, 学习到提示组合的作用机制, 从而帮助生成式模型确定最优的查询执行路径.

提示推荐框架 QueryGuide 的训练过程可分为两个阶段: (1) 单提示学习阶段. 在该阶段, 生成式模型使用一组优化能力较强的查询提示进行参数微调, 学习查询提示的基本优化策略. 微调的目标是使生成式模型理解慢查询的执行特征, 并推荐合理的查询提示. 因此, 微调的数据样本可使用已知的查询提示优化案例, 来帮助生成式模型

掌握提示的应用方式和基本优化规律. (2) 提示组合探索阶段. 微调完成的生成式模型可用于为慢查询推荐单个细粒度的查询提示. 为了使该模型具备更强的提示生成能力, 该阶段对细粒度的提示组合进行探索. 为了提高探索效率, 探索模块基于强化学习的思想, 使用生成式模型推荐的提示作为探索方向, 进行针对性扩展以生成探索提示组合. 通过与 DBMS 进行交互, 可得到探索提示组合所带来的查询性能提升情况, 并以此作为探索样本数据对推荐策略进行迭代优化.

表 1 从 3 个维度展现 QueryGuide 与现有提示推荐方法在推荐机制上的差异. 在提示类型方面, 现有方法针对 6 种布尔类型提示进行探索, 这种粗粒度的优化方式会限制慢查询物理计划的性能提升空间; QueryGuide 首次提出对细粒度提示进行探索, 可以实现对欠优化物理节点的精准调优. 在提示组合策略方面, 现有方法均依赖专家设定候选集, 并基于排序或组合策略进行优选; QueryGuide 首次提出由模型自动生成提示组合, 降低了人工干预程度, 提高了策略组合的灵活性. 在模型作用方式方面, 现有方法采用预测、分类或排序机制对候选提示的性能进行分析评估; QueryGuide 首次提出依据慢查询语句的特性直接生成提示, 实现了从查询特征到提示生成的端到端建模.

表 1　不同提示推荐方法的作用机制

方法	提示类型	提示组合策略	模型作用方式
Bao[11]	布尔	提示组合候选集	打分器
AutoSteer[17]	布尔	单提示候选集自组合	打分器
FASTgres[18]	布尔	单提示候选集自组合	分类预测器
COOOL[16]	布尔	提示组合候选集	得分排序器
Ours	表级别	自生成扩展	生成器

因此, QueryGuide 通过两个阶段的训练, 能够充分结合预训练模型的泛化性学习能力和强化学习的探索能力, 从而生成更优的查询提示组合, 最终提升查询执行性能, 后面的实验结果也证实了这一点.

4　基于生成式模型的细粒度提示推荐策略

4.1　细粒度提示推荐的生成式算法架构

给定一个慢查询, 细粒度的查询提示推荐任务旨在自动生成最优提示, 以提升查询性能. 该任务本质上是一个从慢查询到最优提示的映射过程, 具有显著的生成特性和结构化需求. 为了实现这一目标, 我们引入生成式语言模型, 建模查询与细粒度提示之间的复杂关系来完成提示的自动生成.

为确保生成的提示具备可靠的性能优化能力, 所采用的生成式模型应具备较强的结构化文本生成能力, 能够理解 SQL 查询中的关键语义要素及其对执行计划的影响, 从而生成可直接用于优化的提示. 由于 NLP 领域中 sequence-to-sequence 生成任务 (如机器翻译) 模型可以直接对自然语言进行处理, 而 SQL 与其相比具有更简单且固定的语句结构, 因此可认为, 将细粒度提示推荐建模为现有的生成式语言模型是可行的.

在生成模型的选择上, 我们综合考虑了模型的性能、结构复杂度与部署成本等因素. T5 模型具有相对适中的参数量和成熟的 Encoder-Decoder 双塔结构, 能够保证生成能力的同时降低训练推理成本, 因此本研究中采用 T5 作为实现框架. 具体而言, Encoder 负责深入理解 SQL 语句中的结构化信息, 准确捕捉多表关联、嵌套子查询以及复杂过滤条件等关键特征; Decoder 则专注于生成针对性的细粒度提示, 实现明确的任务分工与逻辑闭环. 此外, T5 模型在超大规模清洁的爬取语料库 (colossal clean crawled corpus, C4) 上进行了预训练, 该语料库包含丰富的网页文本、表格数据及其他半结构化与结构化数据片段, 使得模型具备较强的结构化语言建模能力. 因此, 当 T5 模型迁移到提示推荐任务时, 能够充分发挥其对复杂结构信息的理解能力, 仅需小规模的任务相关数据进行微调即可快速收敛, 生成具有优化价值的细粒度提示. 另外需要指出的是, QueryGuide 框架本身并不依赖具体的生成模型, 在实际应用中可根据需求替换为其他主流大模型.

为完成提示推荐任务, 我们对 T5 模型进行针对性微调, 以充分发挥其对查询语义与性能优化关系的理解和建

模能力. 为此, 我们设计 prompt 为“query”, target 为“hints + *rewards*”的微调数据格式, 如图 3 所示. 其中, query 为慢查询语句, hints 为针对该查询生成的细粒度提示, *rewards* 表示应用该提示后带来的性能收益, 采用执行时间减少量表示.

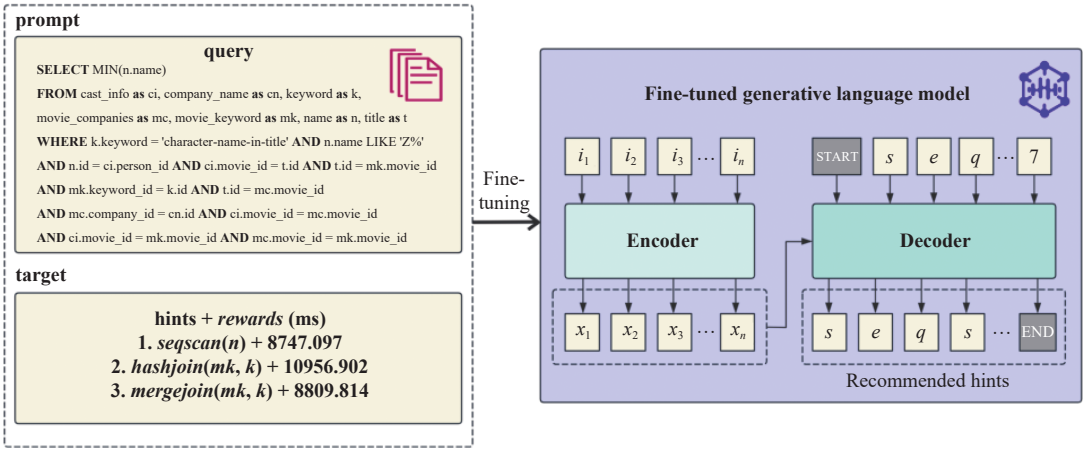

图 3 细粒度查询提示生成模块

这种微调数据设计不仅明确了查询与细粒度提示之间的直接对应关系, 还将性能收益作为显式奖励融入训练过程, 使模型能够学习不同提示对查询性能优化效果的影响. 一方面, T5 模型采用条件语言建模框架, 生成过程由语言模式驱动, 而非简单依赖频率采样. 因此, 其在解码阶段倾向于选择“语言上奖励信号更强”的输出序列. 另一方面, T5 模型在大规模 C4 语料库进行预训练, 使其在学习过程中积累了对因果关系模式、效果描述以及奖励信号表达的强感知能力. 基于此, 微调过程中引入 *rewards* 字段能够进一步强化模型对性能收益显著的提示策略的识别能力与生成偏好, 显著提升生成过程中的收益优化导向性.

4.2 生成式模型微调和提示推荐流程

(1) 模型微调

在数据准备过程中, 为了获取到具有优化价值的细粒度提示作为微调数据, 我们将有效的粗粒度提示作为候选集, 并对其进行细化和重组以生成细粒度提示. 我们首先使用目前生成查询提示的 SOTA 模型 FASTgre, 为慢查询生成有效的粗粒度提示, 然后结合慢查询的语义信息和执行特征来对提示的作用范围进行细化, 将其作为微调数据.

在粗粒度提示生成阶段, 我们主要控制查询计划中的扫描方式 *Scan* 和连接方式 *Join*, 将 FASTgre 模型为慢查询生成的粗粒度提示组合集表示为:

$$CH = \{Scan, Join\},$$

$$Scan = \{enable\_seqscan, enable\_indexscan, enable\_indexonlyscan\},$$

$$Join = \{enable\_hashjoin, enable\_mergejoin, enable\_nestloop\}.$$

各元素的值可设为 on 或 off, 分别表示对应操作的允许或禁用. 这些提示从全局角度指导优化器选择不同的执行策略, 能初步筛选出具有优化效果的物理执行方式.

在查询提示的细化阶段, 运用慢查询 q 中涉及的表集合 T, 可以将粗粒度提示 $c \in CH$ 映射为作用范围更精确、更具针对性的细粒度提示 $f \in FH$. 这是一个一对多的映射过程, 粗粒度提示可转化为多个细粒度提示: CH 中的 $s^c \in Scan$ 操作可转化为 $s^f \in FH$; CH 中的 $j^c \in Join$ 操作可转化为 $j^f \in FH$.

例如 JOB 基准测试中的查询 17b 包含表集合:

$$T = \{ci, cn, k, mc, mk, n, t\}.$$

FASTgre 方法推荐的粗粒度提示为:

$$Scan = \{enable\_indexscan = \text{off}, enable\_seqscan = \text{on}, enable\_indexonlyscan = \text{off}\},$$

$$Join = \{enable\_hashjoin = \text{on}, enable\_mergejoin = \text{on}, enable\_nestloop = \text{off}\},$$

其可转化为细粒度提示候选集:

$$FH = S \cup J,$$

$$S = \{seqscan(t) \mid t \in T\},$$

$$J = \{hashjoin(t_i, t_j), mergejoin(t_i, t_j) \mid t_i, t_j \in T\}.$$

最后, 我们将细粒度提示 $f \in FH$ 运用到慢查询 q 中得到重写语句 q^f. 通过在 DBMS 中执行 q 以及 q^f, 可获取到执行成本 $cost(q)$、$cost(q^f)$, 以及查询性能提升效果:

$$rewards = cost(q) - cost(q^f).$$

采用上述方法生成的样本数据 $(q, f, rewards)$ 对 T5 模型进行微调. 微调的核心目标是使该模型学习到从慢查询 q 映射到最优细粒度提示的能力. 模型需要根据输入的慢查询生成结构化的提示序列, 这本质上属于 Text-to-Text 生成任务. 因此, 微调过程中采用交叉熵损失函数 (cross entropy loss, CEL) 作为优化目标. 对 Decoder 模块生成的 token 序列 $\{o'_1, o'_2, \ldots, o'_{m+1}\}$ 与实际的目标序列 $\{o_1, o_2, \ldots, \text{END}\}$ 计算交叉熵损失:

$$Loss = \frac{1}{m+1}\left(\sum_{i}^{m} CEL(o'_i, o_i) + CEL(o'_{m+1}, \text{END})\right).$$

(2) 模型推理

微调完成的 T5 提示生成模型, 能够为输入的慢查询生成针对性优化的细粒度提示. 将生成的提示运用到慢查询上可构建出重写语句. 将重写语句直接提交至 DBMS 中执行, 以显式方式引导优化器生成性能更优的物理计划, 从而提升慢查询的执行效率.

5 受强化学习启发的迭代式提示组合探索框架

5.1 探索框架

通过使用样本数据对 T5 模型进行微调, 可以使其适应细粒度提示推荐任务, 为慢查询生成具有优化效果的细粒度提示. 由于样本数据使用单个提示进行优化, 因此初步微调得到的 T5 模型无法生成提示组合. 然而, 慢查询的执行计划可能存在多个欠优化的节点, 对其应用提示组合才能最大程度提高执行性能. 因此, 需要进一步探索提示组合的应用方式, 以便 T5 模型可以推荐出最优的提示组合.

受强化学习思想启发, 我们提出一个对查询提示最优组合进行迭代式探索的框架. 通过对 T5 模型生成的提示进行针对性扩展, 并与 DBMS 执行环境交互以探索提示组合的优化效果, 进而使其适应提示组合的推荐任务.

提示组合探索框架包含 3 部分: 生成式 T5 模型、提示扩展模块和 DBMS. 如图 4 所示, 该探索框架的语境与定义 4 相一致: 对于一条查询语句, 生成式模型作为智能体决定探索的方向, 提示扩展模块在该方向的基础上决定探索的动作, DBMS 作为环境决定动作为查询带来的性能收益.

具体来说, 对于慢查询 q, 生成式模型负责推荐细粒度提示 f, 作为探索的整体方向. 提示扩展模块负责在该方向 f 上进行针对性扩展以形成提示组合候选集 FS_q. 按数量渐增的方式进行扩展, 细粒度提示组合的元素数量将从 $|f|$ 扩展到 $|f|+1$. 即, 对细粒度提示候选集使用第 4.2 节中得到的提示候选集 FH 进行扩展:

$$FS_q = FH \cup FS\text{-}extension,$$

$$FS\text{-}extension = \{(f, fh) \mid fh \in FH\}.$$

例如, 对于 JOB 基准测试中的查询 17b, 推荐模型生成提示 $seqscan(n)$, 则:

$$FS\text{-}extension = \{(seqscan(n), fh) \mid fh \in FH\}.$$

将候选集中的元素 $fs \in FS_q$ 分别运用到慢查询 q 上可生成不同的重写语句 q^{fs}. 将其分别在 DBMS 中执行,

可探索出提示组合的性能提升效果.

$$rewards = cost(q) - cost(q^{fs}).$$

$rewards$ 作为探索动作的奖励, 可指导生成式模型进一步优化并适用于提示组合生成任务.

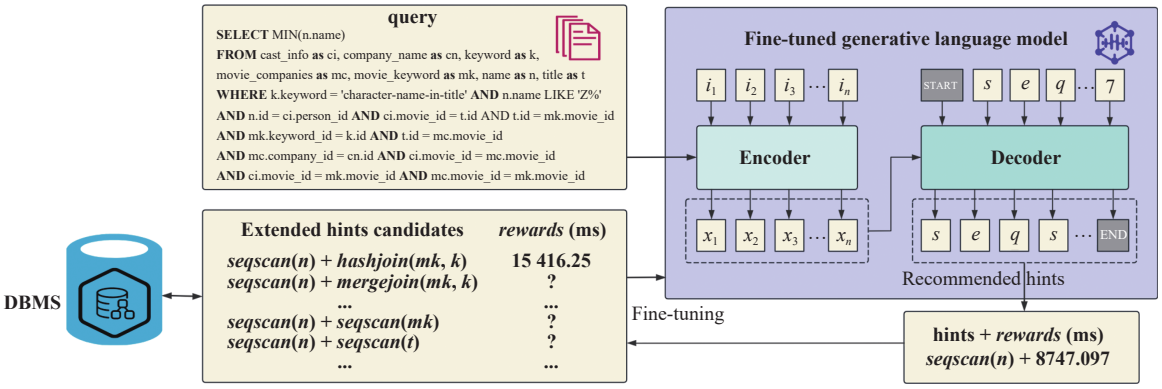

图 4 细粒度提示组合探索模块

5.2 探索流程

给定慢查询 q, 查询提示组合探索算法的整体流程如下, 包含 4 个阶段.

(1) 候选提示生成

使用生成式模型为查询 q 生成初始查询提示 f, 作为后续探索的起点.

(2) 提示扩展

将初始提示 f 输入到提示扩展模块中, 结合细粒度提示候选集 FH, 进行针对性扩展得到提示组合候选集 FS_q.

(3) 提示组合性能探索

将提示组合 $fs \in FS_q$ 添加到查询 q 中形成重写语句 q^{fs}, 并在目标 DBMS 上执行. 通过比较执行成本得到 fs 为查询 q 带来的成本收益 $rewards$.

(4) 模型更新

将探索获得的数据以三元组 $(q, fs, rewards)$ 形式添加到训练数据集中, 并基于新增样本对生成式模型进行迭代式微调.

通过上述 4 个阶段的循环迭代, 模型能够在探索-反馈中动态更新对提示空间的理解, 从而逐步提升提示组合的生成质量.

值得注意的是, 在步骤 (3) 中探索提示组合为查询带来的性能反馈时, 需要在探索成本和精度之间做权衡. 直接在 DBMS 中执行查询能够获得精准的反馈值, 但会消耗时间与计算资源. 因此, 当探索成本显著时, 可以考虑采用当前研究中较为成熟的成本估计器替代实际执行. 相较而言, 该方式用时更短且计算开销更低, 但也会造成一定的精度损失.

6 实验分析

对基于生成式模型的细粒度查询提示推荐方法 QueryGuide 进行重写性能评估, 需要回答以下问题.

(1) 与最先进的提示推荐方法相比, 我们的方法能否提高慢查询执行性能?

(2) 与最先进的提示推荐方法相比, 我们的方法能否提高对工作负载的重写覆盖率?

6.1 数据集与评价指标

实验环境: 我们基于 PostgreSQL12.1 中的 pg_hint_plan 插件进行实验评估, 它同时支持布尔类型的粗粒度提

示以及表级别的细粒度提示. 我们的实验在 128 GB RAM, 2.90 GHz CPU, RTX 3090 GPU, Ubuntu 18.04 的服务器上进行, 在 PyTorch 框架上实现对 T5 模型的微调.

工作负载: 我们使用基于 IMDB 数据库的 JOB 基准测试[20]和 Stack 基准测试[11]来评估我们的方法. IMDB 是一个包含电影信息的真实数据库, 大小为 3.6 GB. Stack 是由 170 个不同的 StackExchange 网站构成的数据库, 大小为 100 GB. 为了评估我们的方法对于实际慢查询场景的有效性, 我们使用 JOB 的全部基准测试以及从 Stack 基准测试中选出的 4 组执行时间较长的工作负载. 表 2 记录了我们所使用的基准测试的执行时间分布情况, 可看出这 5 个工作负载均包含慢查询.

表 2　不同工作负载的执行时间分布情况和查询特征

数据集	执行时间 (s)			查询数量	连接数量
	75^{th}	90^{th}	99^{th}		
JOB	6.51	17.16	149.99	113	4–17
Stack-1	6.00	8.54	190.41	100	4–12
Stack-2	250.86	461.64	759.77	100	4–12
Stack-3	0.81	1.89	13.96	1 008	4–12
Stack-4	1.46	3.65	40.18	1 002	4–12

评估指标: 我们使用以下指标来评估重写方法的性能.

(1) *Latency*: 重写语句的执行时间.

(2) *Latency Reduction*: 重写所带来的查询执行时间减少量:

$$Latency\ Reduction = Latency(slow\ query) - Latency(rewritten\ query).$$

(3) *Coverage*: 工作负载中有效重写的查询数量. 如果重写语句比慢查询的执行时间短, 那么该慢查询所执行的重写是有效的.

(4) *Latency Ratio*: 查询在重写前后的执行时间比率:

$$Latency\ Ratio = Latency(slow\ query)/Latency(rewritten\ query).$$

6.2 对比方法

我们使用下面 3 种经典的查询提示生成方法与 QueryGuide 的性能做比较.

(1) Bao[11]: 使用强化学习的多臂老虎机对查询提示的探索问题进行建模, 将提示组合看作老虎机的一只手臂, 并使用树卷积网络对运用提示的重写语句进行成本预测, 以此选择最优提示组合.

(2) AutoSteer[17]: 使用贪心策略与剪枝策略相结合的方式对查询提示集进行剪枝、合并和探索, 并使用树卷积网络作为运用提示的重写语句的成本估计器.

(3) FASTgres[18]: 使用上下文感知的分类策略, 以连接表和连接属性为依据将慢查询分类, 并为每个查询类别建立基于 XGBoost 模型的最优提示组合预测器.

6.3 重写语句的执行性能分析

为了评估 QueryGuide 生成的查询提示对于提升慢查询性能的有效性, 我们将其与基线方法在 5 个工作负载上进行对比实验.

(1) 查询时间减少量分析

我们比较 QueryGuide 与基线方法为工作负载带来的查询时间减少量. 表 3 和表 4 分别展示了 JOB 和 Stack 工作负载探索的最优提示组合并将其运用到慢查询后, 所带来的查询时间减少量分布. 本文所有表格中, 数据加粗数据表示效果最优, 下划线表示效果次优.

总的来说, QueryGuide 为 JOB 和 Stack 生成查询提示组合, 并运用到慢查询上, 所带来的查询时间减少量在几乎所有指标上都能达到最好的效果. 如在 JOB 和 Stack-1 的 99^{th} 指标上, 我们的方法能为慢查询分别减少约 146 s

和 169 s 的执行时间, 远高于其他方法所带来的减少量. 只有 Stack-3 上 99^{th} 指标略低于 FASTgres, 原因是与其他工作负载相比, Stack-3 的整体查询时间较低, 慢查询数量较少, 因此 QueryGuide 在探索时能够学习到的有效细粒度提示组合较少, 而 FASTgres 在探索时将查询按照执行特点进行分类, 使模型能够学习到更加具有针对性的粗粒度提示, 因此优化效果略好.

表 3　重写语句在 JOB 数据集上的查询时间减少量 (s)

方法	75^{th}	90^{th}	99^{th}	mean
FASTgres	0.14	8.25	143.12	7.00
Bao	0.00	2.11	18.42	1.25
AutoSteer	0.08	4.79	41.89	2.84
Ours	**0.63**	**12.50**	**145.95**	**7.62**

表 4　重写语句在 Stack 数据集上的查询时间减少量 (s)

方法	Stack-1			Stack-2 ($\times 10^3$)			Stack-3			Stack-4		
	75^{th}	99^{th}	mean	75^{th}	99^{th}	mean	75^{th}	99^{th}	mean	75^{th}	99^{th}	mean
FASTgres	0.04	1.38	0.06	0.14	0.73	0.11	0.06	**1.68**	**0.11**	0.04	1.23	0.11
Bao	0.02	0.71	0.03	0.02	0.36	0.05	0.03	0.87	0.06	0.02	0.64	0.06
AutoSteer	0.02	60.06	54.43	0.07	0.23	0.05	0.03	0.53	0.04	0.03	1.59	0.08
Ours	**0.06**	**168.72**	**167.51**	**0.25**	**0.76**	**0.15**	**0.08**	1.49	**0.11**	**0.09**	**4.46**	**0.22**

综合来看, QueryGuide 使用生成式模型探索细粒度提示的组合, 并通过微调的方式学习到不同提示组合为查询带来的性能提升效果, 与粗粒度提示推荐算法相比, 我们的方法能够为慢查询带来更大的性能提升.

(2) 重写语句执行时间分析

我们比较 QueryGuide 与基线方法生成重写语句的执行时间. 图 5 和图 6 分别展示了为 JOB 和 Stack 工作负载探索最优提示组合并运用到慢查询上, 得到的查询时间分布. 由于工作负载内部各查询的执行时间差异较大, 我们采用了两张具有不同坐标刻度的图, 分别标识为 A、B, 以便更清晰地展示各查询的执行时间. 图中的横坐标为工作负载内各查询按照执行时间从小到大排列的次序.

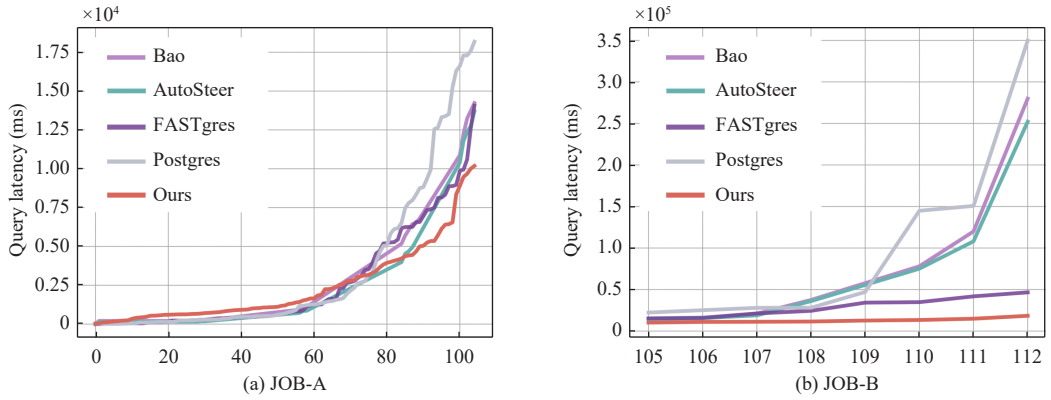

图 5　JOB 数据集上重写语句的查询时间分布

图 5(a) 和图 5(b) 展示出 JOB 工作负载运用提示组合后的查询时间. 显然, 与其他方法相比, QueryGuide 的尾部时延更短, 能够为慢查询带来更显著的性能提升. 这是因为 QueryGuide 在探索时, 适用于慢查询的细粒度提示组合能够为其带来更大的奖励值, 因此为其分配的探索权重更高. 然而, 对于按查询时间排序的前 70 条查询而言, 因其执行时间在 1.5 s 之内, 优化空间相对较小, 因此 QueryGuide 对这部分查询的性能提升相对有限.

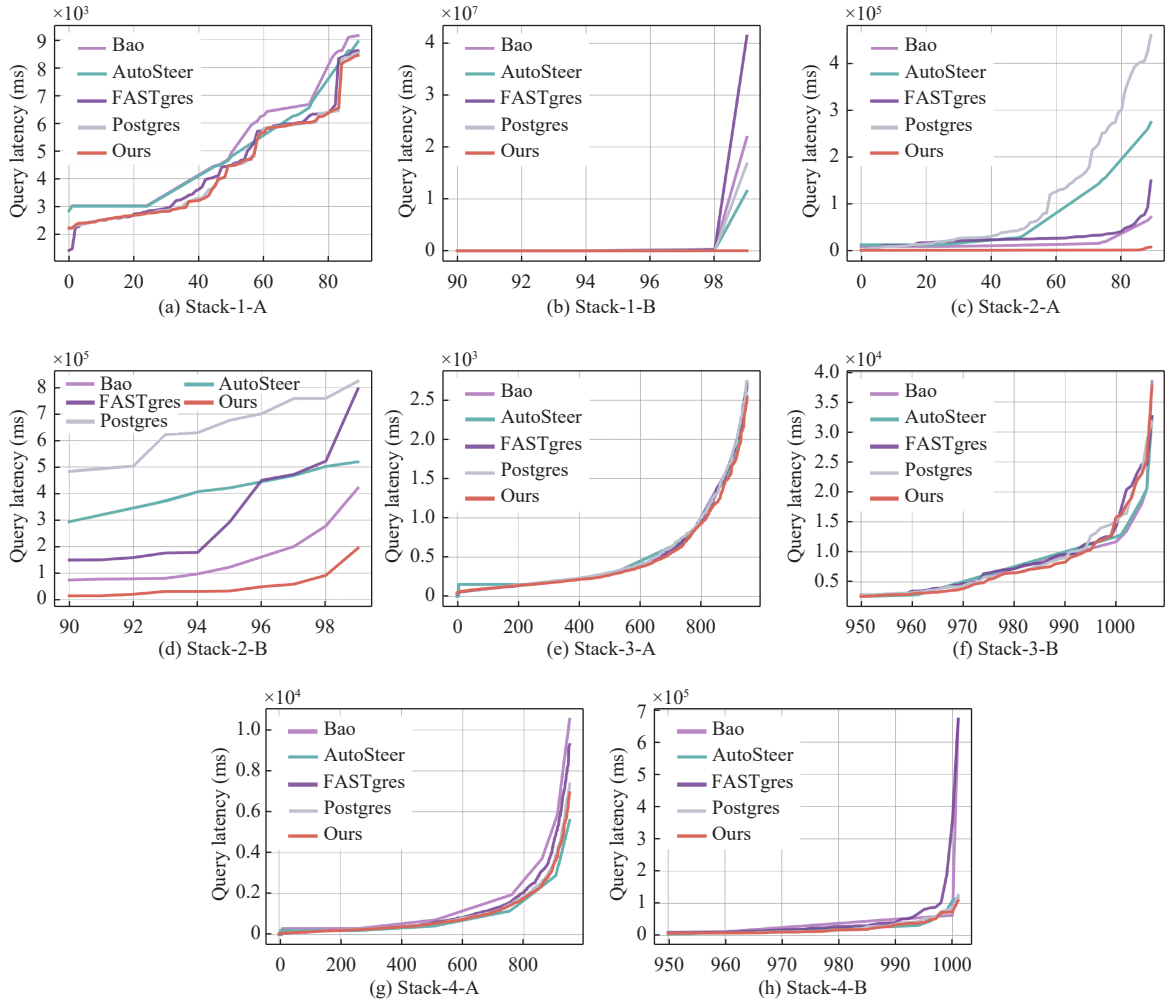

图 6　Stack 数据集上重写语句的查询时间分布

图 6(a) 和图 6(b) 展示出 Stack-1 运用提示组合后的查询时间. 显然, 与其他方法相比, QueryGuide 能为大多数查询实现更短的执行时间, 同时性能退化的情况最少, 而且能为时间最长的查询减少约 2.8 h 的执行时间. 这是因为 QueryGuide 生成的细粒度提示仅对执行计划进行局部调整: 一方面, 能够针对欠优化节点进行精准调优, 从而提升查询性能; 另一方面, 由于调整幅度有限, 即使提示的定位存在偏差, 也不易造成整体性能的显著退化. 图 6(c) 和图 6(d) 展示出 Stack-2 运用提示组合后的查询时间. 显然, 与其他方法相比, QueryGuide 在所有查询上均实现了更短的执行时间, 尤其是在处理时间最长的查询时, 相较于次优方法, 查询时间进一步减少了 200 s 左右. 这是因为工作负载 Stack-2 的执行时间较长, 查询计划中存在较多未被充分优化的节点, QueryGuide 能生成针对性强的细粒度提示, 进而显著降低查询时间. 图 6(e)–(h) 展示出 Stack-3 和 Stack-4 运用提示组合后的查询时间. 可以看出, 尽管这两个工作负载的查询时间相对较短, 但 QueryGuide 引发的性能退化情况很少, 表明其生成的提示对查询计划的性能具有较低的优化风险. 这主要得益于在探索过程中, QueryGuide 对细粒度提示的效果持续学习和评估, 因此能够有效规避可能造成性能下降的提示.

6.4　工作负载的重写覆盖情况分析

我们进一步比较 QueryGuide 与基线方法在重写覆盖数量和慢查询语句性能提升方面的效果.

表 5 展现了为 5 个工作负载生成细粒度查询提示, 并将其运用到慢查询上, 得到的有效重写数量. 显然, QueryGuide 在大多数工作负载上都能生成数量最多的有效查询提示, 特别是对于慢查询数量最多的 Stack-2, 能够比次优方法多生成 36 个有效的查询提示. 另一方面, 对于表连接数量较多的 JOB, 我们的方法生成的有效提示数量略低于 FASTgres. 这是因为 FASTgres 将查询按照表连接方式进行分类探索, 能对连接数量多的工作负载实施一定程度的普适性优化, 而我们的方法更擅长对查询计划的缓慢部分进行定位以及对慢查询进行精准优化.

表 5 不同数据集上有效重写语句的数量

方法	JOB	Stack-1	Stack-2	Stack-3	Stack-4
FASTgres	**37**	**28**	59	465	388
Bao	23	24	71	371	312
AutoSteer	28	24	45	374	409
Ours	34	**28**	**95**	**483**	**411**

图 7 和图 8 展现了 QueryGuide 与基线方法为 5 个工作负载生成查询提示, 并将其运用在慢查询上, 统计得到的查询时延率分布情况. 在图 7 和图 8 中, 我们将有效的重写语句按照查询时延率进行划分. 查询时延率越高颜色越深, 说明重写方法带来的性能提升越大.

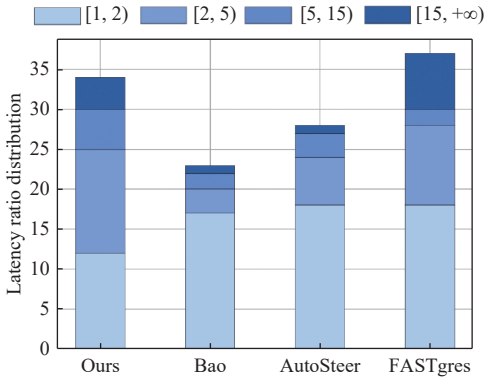

图 7 JOB 数据集上有效重写语句的时延率分布

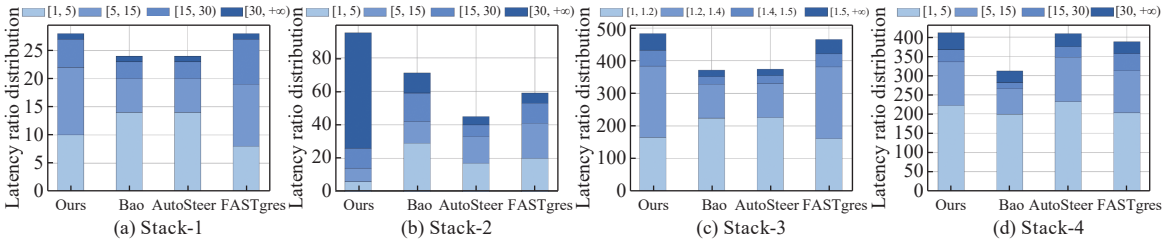

图 8 Stack 数据集上有效重写语句的时延率分布

图 7 展示出在 JOB 上运用提示组合后得到的查询时延率. 可以看出, QueryGuide 所生成的有效提示组合中, 带来高查询时延率的提示组合占比更大. 这是因为我们的方法采用细粒度的提示, 能够面向慢查询进行精准调优. 图 8(a) 展示出在 Stack-1 上运用提示组合后得到的查询时延率. 可以看出, QueryGuide 的整体性能与 FASTgres 相当, 仅在时延率分布在 15–30 区间的提示上略低. 这是因为工作负载 Stack-1 只包含 100 条查询, 在生成式模型参数规模较大的情况下, 限制了其学习效果. 实际上, QueryGuide 的微调时间远短于 FASTgres 的训练时间, 表明该方法在实际部署中更具有操作性. 图 8(b) 展示出在 Stack-2 上运用提示组合后得到的查询时延率. 显然, 相较于其他方法, QueryGuide 为该工作负载生成的提示带来了整体更高的时延率, 其中有 71 条提示对应的时延率超过 30. 这是因为 Stack-2 中的查询整体执行缓慢、性能瓶颈较多, 而 QueryGuide 通过学习不同提示的奖励来识别查询计

划的性能瓶颈, 因此能够精准定位欠优化节点并改进其物理执行方式. 图 8(c) 和图 8(d) 分别展现出在 Stack-3 和 Stack-4 上运用提示组合后得到的查询时延率. 可以看出, 相较于其他方法, QueryGuide 在整体时间较短的工作负载上仍能生成更高时延率的提示. 这是因为这两个工作负载的查询数量较多, 使生成式模型充分学习到查询的执行模式及其潜在的性能瓶颈, 从而在优化空间较小的查询计划上亦能表现出良好的提示推荐能力.

总的来说, 我们的方法能够为慢查询生成更加有效的提示组合, 以提升重写语句的执行效率. 一方面, 我们使用细粒度查询提示引导查询计划节点的操作方式, 实现更精准的优化控制. 另一方面, 我们运用强化学习思想对提示组合的优化水平进行探索, 并以性能提升程度作为模型训练的奖励信号, 从而充分挖掘慢查询的优化潜力.

6.5 提示推荐策略有效性评估

我们分别评估 QueryGuide 的两种提示推荐策略对提升慢查询重写性能的有效性.

(1) 微调数据格式的有效性评估

我们分析采用 (SQL, hints + *rewards*) 数据格式进行微调对于提升 QueryGuide 推荐性能的有效性.

对于生成式模型的微调格式设计, 当前的查询优化工作 IdxL[21]提出了查询的格式化方法, 将查询中的表和列分别表示为 t_i 和 c_i 的形式. 我们使用以下 5 种不同的微调数据格式来为 JOB 生成细粒度的查询提示, 将它们运用到慢查询上, 并分析它们对查询执行时间的影响.

1) 采用 (fmt_SQL, hints) 数据格式: prompt 为表和列经过格式化的查询语句, label 为对应查询的最优细粒度提示组合.

2) 采用 (fmt_SQL, fmt_hints) 数据格式: prompt 为表和列经过格式化的查询语句, label 为同样经过格式化的最优细粒度提示组合.

3) 采用 (SQL, hints) 数据格式: prompt 为原始的查询语句, label 为对应查询的最优细粒度提示组合.

4) 采用 (SQL + *rewards*, hints) 数据格式: prompt 为原始查询语句与期望性能提升水平的组合, label 为能达到该期望水平的细粒度提示组合.

5) 采用 (SQL, hints + *rewards*) 数据格式: prompt 为原始的查询语句, label 为细粒度提示与其能达到的性能提升水平的组合.

表 6 展示了 QueryGuide 分别采用上述 5 种数据格式微调后, 其推荐的提示应用在 JOB 上所得到的重写语句执行时间. 为了对比效果, 表中还列出了慢查询在 PostgreSQL 上的原始执行时间. 显然, 与其他格式相比, (SQL, hints+*rewards*) 数据格式能为慢查询带来更低的执行时间. 特别是在 99th 和 max 指标上, 采用该数据格式能使慢查询分别减少 135 s 和 331 s 的执行时间, 与次优的数据格式相比, 分别降低约 2.5 s 和 36 s 的时间. 实际上, 直接采用原始查询语句 SQL 及其提示组合 hints 进行模型微调, 能够有效捕捉优化器对特定数据库的欠优化部分, 从而使生成的查询提示对慢查询性能的提升更有针对性. 另一方面, 在训练数据中标注提示组合的性能提升收益 *rewards*, 可使模型以显式奖励方式学习不同提示组合的优化效果. 相较于仅学习最优提示组合, 这种监督方式通过提供更丰富的优化知识, 显著增强了模型的泛化能力, 最终实现更优的性能提升效果.

表 6 不同微调数据格式作用在 JOB 数据集上得到的重写语句执行时间分布 (s)

方法	25th	50th	75th	90th	99th	mean	max	min
PostgreSQL	0.25	1.17	6.51	17.16	149.99	10.15	349.85	**0.03**
(fmt_SQL, hints)	0.25	0.74	5.21	17.39	162.16	8.17	184.53	0.04
(fmt_SQL, fmt_ hints)	**0.22**	**0.73**	5.32	21.39	162.19	8.88	183.55	0.04
(SQL, hints)	0.68	1.53	3.90	10.12	17.37	3.69	54.99	0.05
(SQL+*rewards*, hints)	0.78	1.46	**3.61**	9.99	50.22	5.10	189.03	0.04
(SQL, hints+*rewards*)	0.66	1.43	4.22	**9.36**	**14.70**	**3.14**	**18.41**	0.04

(2) 细粒度提示组合的有效性评估

我们分析探索细粒度提示组合对于提升 QueryGuide 重写性能的有效性. 通过使用 QueryGuide 来为 JOB 分别探索包含 4 种不同数量查询提示的组合, 并将其运用在慢查询上, 来分析它们对于重写语句执行时间的影响.

1) 单提示组: 探索单一操作的查询提示, 允许 1 个扫描方式或 1 个连接方式.

2) 双提示组: 探索包含两个操作的提示组合, 可以包括 1 个扫描方式+1 个连接方式、2 个扫描方式、2 个连接方式.

3) 三提示组: 探索包含 3 个操作的提示组合, 可以包括 3 个扫描方式、1 个扫描方式+2 个连接方式、2 个扫描方式+1 个连接方式.

4) 四提示组: 探索包含 4 个操作的提示组合, 可以包括 2 个扫描方式+2 个连接方式、3 个扫描方式+1 个连接方式、4 个扫描方式.

图 9 展示了 QueryGuide 分别探索上述 4 种提示方式, 并运用到 JOB 上所得到的重写语句执行时间分布情况. 图 9(a) 和图 9(b) 分别展示了工作负载 JOB 的前 55 条查询和后 58 条查询运用不同提示类别后的执行时间. 显然, 多数查询采用提示组合优化后均能实现最短的执行时间, 且在某些情况下, 提示组合与单个细粒度提示相比会更显著地降低查询时间. 如第 8 个查询采用包含 1 个扫描方式+2 个连接方式的组合、第 16 个查询采用包含 4 个扫描方式的组合、第 88 个查询采用 1 个扫描方式+2 个连接方式的组合后均能实现最低的查询时间, 与单个提示相比, 能分别减少约 4 s、5 s、7 s 的时间. 这是因为多个细粒度提示的组合能够为执行计划的不同节点进行精准调优, 因此能够带来的优化效果也更为显著. 实验结果表明, 探索细粒度的提示组合对于慢查询提速是很有必要的.

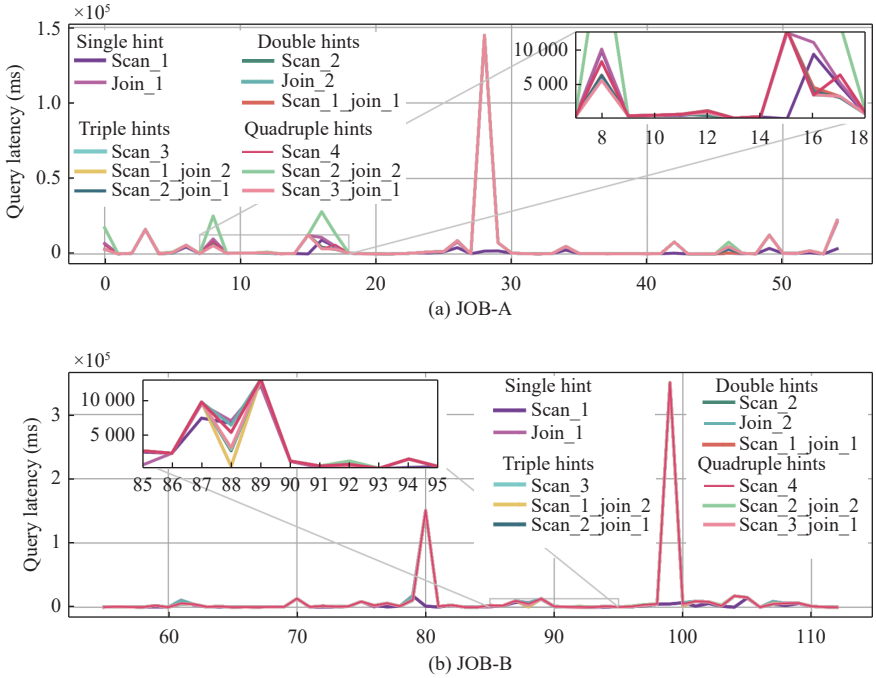

图 9 不同查询提示组合作用在 JOB 数据集上得到的重写语句执行时间

6.6 生成式模型可替换性分析

为分析细粒度提示生成框架中 T5 模型的可替换性, 我们选取多个主流生成式大语言模型. 通过采用统一的交互式 prompt 策略, 引导模型学习历史优化经验, 并自动为慢查询生成细粒度提示. 相较于 T5 模型, 大语言模型在预训练阶段使用更大规模的语料和参数量, 具备更强的语义理解和知识迁移能力. 因此本实验仅使用少量优化经验进行引导, 使模型在快速适应提示生成任务的同时, 能够充分利用自身具备的通用知识.

表 7 展示不同大语言模型在 JOB 工作负载上生成提示, 得到的重写语句执行时间分布. 实验结果表明, 所有大模型均能取得不同程度的性能提升, 表明提示推荐框架中生成式模型具有可替换性. 具体来说, Grok 4 与 DeepSeek-V3 在平均运行时间及高分位数指标上表现更优, 显著降低了查询尾部延迟. 这是因为其生成的提示结

构与历史经验更为一致, 策略相对稳健. 相比之下, GPT-5-mini 与 Kimi K2 倾向于生成结构较为激进的提示, 如历史经验中未出现的 Leading 类型提示, 因此工作负载会存在长尾效应. 这也表明, 当历史经验中增加更多 Leading 类型的提示时, 其性能会具有更大的提升空间.

表 7　不同生成式模型的参数量以及作用在 JOB 数据集上得到的重写语句执行时间分布

方法	参数规模	查询时间分布 (s)			
		75^{th}	90^{th}	99^{th}	mean
GPT-5-mini	约149B	6.12	10.90	148.49	8.92
DeepSeek-V3	671B	4.10	9.61	18.99	3.11
Kimi K2	1T	**4.07**	10.66	132.01	7.45
Grok 4	约1.7T	4.16	**7.00**	16.81	**3.01**
T5	**220M**	4.22	9.36	**14.70**	3.14
Baseline (no hints)	—	6.51	17.02	148.00	10.09

另一方面, 从表 7 可看出, T5 的参数规模比主流大语言模型小 3–4 个数量级, 但在 99^{th} 分位数指标上仍取得最好表现, 显示出更高的尾部稳定性. 结合前文实验结果, 不难看出 T5 能够有效适配提示推荐任务. 与此同时, 从实际部署角度考虑, T5 具备更低的计算与存储开销成本, 更适合本地化部署与微调训练. 因此, 该模型在工程实践中具有较高的经济性与可行性.

6.7　模型可迁移性分析

不同数据库管理系统在查询优化器结构、代价模型以及支持的查询提示类别方面均存在差异, 因此提示的适用条件及其性能收益会因系统而异. 而 QueryGuide 的生成式模型本质上会学习并拟合目标数据库管理系统所支持的提示空间, 在某个数据库管理系统上训练得到的模型, 其输出会自然偏向该体系下的提示类别, 而这些提示在其他系统中可能不存在, 或其对应的执行代价存在显著差异. 因此, 尽管 QueryGuide 框架的方法论是跨系统可复用的, 但具体的提示生成模型通常无法直接迁移. 针对不同的目标数据库管理系统, 需要根据提示体系重新设计数据采集流程并训练模型, 使学到的语义适配新的提示空间.

然而, 如果两个数据库管理系统有相似的提示体系, 如 Oracle 与 PostgreSQL, 它们均包含单表访问策略 FULL、INDEX、NO_INDEX, 以及多表连接方式 USE_NL、USE_MERGE、USE_HASH. 因此, PostgreSQL 上的数据收集与模型训练流程可以以较低的成本迁移至 Oracle, 仅需对标签进行调整. 相比之下, MySQL 与 PostgreSQL 的提示体系之间存在较大差异. MySQL 的优化器控制提示同时包含粗粒度的全局开关, 如对 batched_key_access、block_nested_loop、derived_merge 的启用与禁用, 以及基于指定表的细粒度提示, 如 BKA(t)/No_BKA(t)、BNL(t1, t2)/No_BNL(t1, t2)、Merge(t(, t2))/No_Merge(t(, t2)). 这些提示在 PostgreSQL 中不存在对应特征, 因此需要从提示体系定义开始重新构建提示空间, 并独立开展数据收集与模型训练.

7　总　结

本文提出 QueryGuide, 一个受强化学习启发的迭代式细粒度查询提示推荐框架. 通过探索不同查询提示组合并利用探索结果对生成式语言模型进行微调, 从而使模型能够生成细粒度的提示组合, 进而提升重写语句的执行效率. 我们在 PostgreSQL 上针对多个慢查询负载进行了广泛的实验, 实验结果验证了该方法的有效性. 它能够为慢查询生成合适的细粒度查询提示, 为欠优化的计划节点选择出最优的物理执行方式, 从而提高慢查询的执行效率.

References

[1] Chen J, Jindel S, Walzer R, Sen R, Jimsheleishvilli N, Andrews M. The MemSQL query optimizer: A modern optimizer for real-time analytics in a distributed database. Proc. of the VLDB Endowment, 2016, 9(13): 1401–1412. [doi: 10.14778/3007263.3007277]

[2] Zhou XH, Chai CL, Li GL, Sun J. Database meets artificial intelligence: A survey. IEEE Trans. on Knowledge and Data Engineering, 2022, 34(3): 1096–1116. [doi: 10.1109/TKDE.2020.2994641]

[3] Wang ZG, Zhou Z, Yang YC, Ding HR, Hu GS, Ding D, Tang CZ, Chen HB, Li JY. WeTune: Automatic discovery and verification of

query rewrite rules. In: Proc. of the 2022 Int'l Conf. on Management of Data. Philadelphia: ACM, 2022. 94–107. [doi: 10.1145/3514221.3526125]

[4] Zhou Q, Arulraj J, Navathe S, Harris W, Wu J P. Sia: Optimizing queries using learned predicates. In: Proc. of the 2021 Int'l Conf. on Management of Data. New York: ACM, 2021. 2169–2181. [doi: 10.1145/3448016.3457262]

[5] Wang YR, Khamis MA, Ngo HQ, Pichler R, Suciu D. Optimizing recursive queries with progam synthesis. In: Proc. of the 2022 Int'l Conf. on Management of Data. Philadelphia: ACM, 2022. 79–93. [doi: 10.1145/3514221.3517827]

[6] Zhou XH, Li GL, Chai CL, Feng JH. A learned query rewrite system using Monte Carlo tree search. Proc. of the VLDB Endowment, 2021, 15(1): 46–58. [doi: 10.14778/3485450.3485456]

[7] Graefe G, McKenna WJ. The Volcano optimizer generator: Extensibility and efficient search. In: Proc. of the 9th In'l Conf. on Data Engineering. Vienna: IEEE Computer Society, 1993. 209–218. [doi: 10.1109/icde.1993.344061]

[8] Graefe G, DeWitt D J. The EXODUS optimizer generator. In: Proc. of the 1987 ACM SIGMOD Int'l Conf. on Management of Data. San Francisco: ACM, 1987. 160–172. [doi: 10.1145/38713.38734]

[9] Graefe G. The cascades framework for query optimization. IEEE Database Engineering Bulletin, 1995, 18(3): 19–29. [doi: 10.1109/ride.1992.227411]

[10] Li YY, Tian JK, Pu Z, Li CP, Chen H. Research survey on intelligent tuning of database knobs configuration. Chinese Journal of Computers, 2024, 47(8): 1901–1921 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2024.01901]

[11] Marcus R, Negi P, Mao HZ, Tatbul N, Alizadeh M, Kraska T. Bao: Making learned query optimization practical. In: Proc. of the 2021 Int'l Conf. on Management of Data. New York: ACM, 2021. 1275–1288. [doi: 10.1145/3448016.3452838]

[12] Dong R, Liu J, Zhu YX, Yan C, Mozafari B, Wang XY. SlabCity: Whole-query optimization using program synthesis. Proc. of the VLDB Endowment, 2023, 16(11): 3151–3164. [doi: 10.14778/3611479.3611515]

[13] Yu X, Chai CL, Zhang XN, Tang N, Sun J, Li GL. AlphaQO: Robust learned query optimizer. Ruan Jian Xue Bao/Journal of Software, 2022, 33(3): 814–831 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6452.htm [doi: 10.13328/j.cnki.jos.006452]

[14] Bai QS, Alsudais S, Li C, Zhao S. Maliva: Using machine learning to rewrite visualization queries under time constraints. In: Proc. of the 26th Int'l Conf. on Extending Database Technology. Ioannina: OpenProceedings.org, 2023. 157–170. [doi: 10.48786/edbt.2023.13]

[15] Yu X, Chai CL, Li GL, Liu JB. Cost-based or learning-based?: A hybrid query optimizer for query plan selection. Proc. of the VLDB Endowment, 2022, 15(13): 3924–3936. [doi: 10.14778/3565838.3565846]

[16] Xu XH, Zhao ZB, Zhang TY, Kang R, Sun LM, Chen JJ. COOOL: A learning-to-rank approach for SQL hint recommendations. In: Proc. of Workshops at the 49th Int'l Conf. on Very Large Data Bases. Vancouver: CEUR-WS.org, 2023. 1–16.

[17] Anneser C, Tatbul N, Cohen D, Xu ZG, Pandian P, Laptev N, Marcus R. AutoSteer: Learned query optimization for any SQL database. Proc. of the VLDB Endowment, 2023, 16(12): 3515–3527. [doi: 10.14778/3611540.3611544]

[18] Woltmann L, Thiessat J, Hartmann C, Habich D, Lehner W. FASTgres: Making learned query optimizer hinting effective. Proc. of the VLDB Endowment, 2023, 16(11): 3310–3322. [doi: 10.14778/3611479.3611528]

[19] Raffel C, Shazeer N, Roberts A, Lee K, Narang S, Matena M, Zhou YQ, Li W, Liu PJ. Exploring the limits of transfer learning with a unified text-to-text Transformer. Journal of Machine Learning Research, 2020, 21(140): 1–67.

[20] Leis V, Gubichev A, Mirchev A, Boncz P, Kemper A, Neumann T. How good are query optimizers, really? Proc. of the VLDB Endowment, 2015, 9(3): 204–215. [doi: 10.14778/2850583.2850594]

[21] Peng G, Cai P, Ye KK, Li K, Cai JL, Shen YF, Su H, Xu WY. Online index recommendation for slow queries. In: Proc. of the 40th IEEE Int'l Conf. on Data Engineering. Utrecht: IEEE, 2024. 5294–5306. [doi: 10.1109/ICDE60146.2024.00398]

附中文参考文献

[10] 李奕言, 田季坤, 蒲照, 李翠平, 陈红. 数据库参数配置智能调优研究综述. 计算机学报, 2024, 47(8): 1901–1921. [doi: 10.11897/SP.J.1016.2024.01901]

[13] 余翔, 柴成亮, 张辛宁, 汤南, 孙佶, 李国良. AlphaQO: 鲁棒的学习型查询优化器. 软件学报, 2022, 33(3): 814–831. http://www.jos.org.cn/1000–9825/6452.htm [doi: 10.13328/j.cnki.jos.006452]

作者简介

杨少聪, 硕士, CCF 学生会员, 主要研究领域为慢查询重写, 查询优化, 强化学习.

王宁, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为人工智能赋能的数据管理, 大数据与群智计算, 数据挖掘.