arXiv:2602.08546v1 [cs.DB] 9 Feb 2026

AnalyzeOperator 的语义与多查询优化算法

Marios Iakovidis [0009−0006−9061−3450]约阿尼纳大学，约阿尼纳 45110，希腊 miakovidis@uoi.gr

Panos Vassiliadis [0000−0003−0085−6776]约阿尼纳大学，约阿尼纳 45110，希腊 pvassil@cs.uoi.gr

2026年2月10日

本文的一个简短版本已被 DOLAP 2026 接收：M. Iakovidis、P. Vassiliadis。《面向新型 Analyze 算子的多查询优化》。已被第 28 届国际大数据设计、优化、语言与分析处理研讨会（DOLAP）接收。该研讨会与 EDBT/ICDT 2026 在芬兰坦佩雷同期举行，日期为 2026年3月24日。

摘要

在寻找亮点（即数据中有趣的模式）时，数据分析师必须发出 *groups* 个相关查询，并 *manually* 合并其结果。如果分析师的目标基于对发现内容的 *inten-tion*（例如，将查询结果与同级查询结果进行对比，验证某种模式是否适用于数据空间中范围更广的数据等），那么在分析引擎中集成 *intentional* 查询算子便可提升这些分析任务的效率。本文将介绍一种新颖的立方体查询意向算子—— *ANALYZE operator*，并为其定义明确的语义。该算子能够提供数据的 360^o 视图。我们将 ANALYZE 查询的语义定义为五个内部辅助立方体查询构成的元组，这些查询：(a) 报告查询规范中指定的、数据空间中特定子集的详细信息，我们将其称为*original query*；(b) 将结果与来自同级子空间（即 *sibling queries*）的结果进行对比；(c) 通过 *drill-down queries*以更低粒度层级探索数据空间。我们为该算子提出了形式化的查询语义，并从理论上证明，将辅助立方体查询合并为数量更少的查询，也能得到完全相同的结果。这实际上为执行 ANALYZE 查询引入了一种 *multi-query optimization (MQO)*策略。我们提出了三种替代算法：(a) 不进行优化的简单执行（*Min-MQO*）；(b) 将所有辅助查询完全合并为一个查询（*Max-MQO*）；

以及 (c) 一种中间策略，即 *Mid-MQO*，它只合并部分辅助查询。实验结果表明，Mid-MQO 在多种场景下始终表现出色，Min-MQO 的表现始终紧随其后，而对于兄弟查询规模较大且重叠显著的情况，Max-MQO 则表现突出。

1 引言

数据分析师通常会发出成组的相关查询，以期从数据中发掘隐藏的宝藏。这些重要发现，即亮点，实际上验证了感兴趣的数据子集中存在有趣属性（例如趋势、峰值、异常值等）。因此，自动提取亮点是加快数据分析师工作、实现对数据空间更广泛且全面探索的重要任务。如果分析师的目标基于对发现内容的意图（例如，将查询结果与探索数据空间中与当前审查对象相似的子集所得的查询结果进行对比；验证某种模式是否适用于更广泛的数据范围；预测趋势等），我们便需要意图驱动的查询算子 [VM18, VMR19]，将这些数据探索抽象为更高层次的算子。因此，这些算子应具备数据的形式化模型、形式语义，当然还应有用于执行它们的优化方法。

关于自动亮点提取的相关研究提供了用于多维数据探索的工具，以发现有趣的数据模式 [Sar99, SAM98, WSZ+20, AKS+21, MDHZ21]。这些工具主要致力于探索数据空间、发现数据模式，并使用趣味性度量指标衡量它们的有趣程度。然而，研究现状呈现出以下特点：(a) 缺乏一种广受认可的形式化模型，能够在统一框架中涵盖所提出的任务（ [VM18, VMR19] 的意图模型是首次尝试）；(b) 总体而言，忽视了利用多维分层数据空间（按商业智能领域的含义）的可能性，研究几乎完全聚焦于关系数据；(c) 据我们所知，尚无一种算子能够通过单次调用，为分析师提供全面的数据视图。

本文提出了 *ANALYZE* 运算符，这是一种新颖的立方体查询运算符，通过一次调用，将向数据分析师提供 360^o 数据视图的意图形式化。作为一种代数算子，ANALYZE 具备：(a) 定义明确的多维层次底层数据模型，其基本元素为数据立方体和维度 [Vas22], (b) 对预期结果有定义明确的语义；此外，在本文中还包括 (c) 用于高效执行的优化策略。层次多维模型的简洁性使查询算子也得以简化，并能充分利用该模型的能力，在不同粗粒度层级上顺畅地计算兄弟、祖先和后代值。为便于理解，本文将以下查询作为贯穿全文的参考示例：

ANALYZE	$sum(Store\ Sales)$ 作为 $SumSales$
FROM	$Sales$
FOR	$Date.Quarter\ =\ 1997-Q3\ \wedge$ $Customer.state\ =\ 'CA'\ \wedge$ $Promo.Media\ =\ 'Daily\ Paper'$
GROUP BY	$month,\ customerRegion$
AS	$PaperPromoCA1997Q3$

ANALYZE 查询的语义包含五个内部的协调器立方体查询。首先，算子必须处理针对数据空间中一个非常具体子集所请求的信息；该子集通过 (a) 一组过滤器和 (b) 一对分组器获得，后者以 GROUP-BY 的方式用于聚合某项度量。我们将其称为算子的原始查询，因为它针对的是感兴趣的数据空间子集。在此示例中，对 1997年第三季度、加利福尼亚州和《日报》设置的三个过滤条件，将涉及的数据限制在感兴趣的子空间内。结果按月份和客户地区对数据分组，并报告门店销售额的总和。其次，我们希望将结果与相似的同类结果进行对比。为此，我们结合分组器和过滤条件，获取数据中的兄弟子空间；这些查询子空间与原始查询的过滤条件共享相同的祖先值。在本例中，由于我们希望将时间限定为 1997年第三季度，因此需要找到与 1997年第三季度相似的季度，以便对比结果。这时，多维空间的层次结构便发挥了作用：它使我们能够将共享相同祖先值的值定义为兄弟。假设我们按年对季度进行分组，就可以将原始查询与 1997年的各季度进行对比，因为 1997年第三季度属于 1997年。同样，为了找到加利福尼亚州的同类项，我们将其与美国的其他州进行对比。最后，我们通过所谓的下钻查询（与传统联机分析处理中的含义基本相同），在更低粒度层级上探索数据空间，并报告 1997年第三季度各月份以及居住在加利福尼亚州的客户的门店销售额总和。我们在此强调，重点在于代数算子本身，而非语言层面：ANALYZE 查询可以通过命令行界面调用、类SQL表达式或点击式界面生成，但最终都会执行同一个代数算子。

然而，除了为该算子引入形式化的查询语义之外，我们还从理论上证明，通过将辅助立方体查询合并为数量更少的合并查询，并利用立方体可用性结果减少冗余计算，我们能够获得完全相同的结果。这一理论结果使我们得以引入一种多查询优化（*MQO*）策略，将多个协作查询合并为一个或多个合并查询，以尝试加快 ANALYZE 查询的执行速度。事实上，我们设计了多种多查询优化策略，用于将底层辅助查询合并为一个查询。在最简单的情况下，不进行优化，五个辅助查询彼此独立地执行：我们称之为$Min\text{-}MQO$，因为它引入的查询合并程度最低。其次，我们遵循这一理论结果，将所有

辅助查询合并为一个查询，这一策略称为*Max-MQO*，因为它实现了最初提出的将所有查询合并的理论结果。最后，根据我们最初的实验观察（这些观察发现上述策略在某些情况下性能较低），我们还引入了一种折中的策略*Mid-MQO*，在两个极端之间取得平衡，只合并部分辅助查询。在所有合并策略中，获取合并查询的结果后，都会对其进行后处理，以正确填充各个辅助查询。

我们通过多个数据集上的各种查询工作负载，对每种多查询优化策略的性能进行了广泛评估，以考察数据规模和查询选择率对执行成本的影响以及最优策略。我们的实验表明，*Mid-MQO* 是最高效的算法，能够随数据规模和选择率平稳扩展，并且通常取得最佳性能。尽管在许多情况下*Max-MQO* 性能最差，但当兄弟查询规模较大且显著重叠时，*Max-MQO* 是最快的算法；我们可以通过决策树预测何时会出现这种情况。最后，*Min-MQO* 从来都不是最优策略，尽管它与*Mid-MQO*相当接近。

本文余下部分结构如下。第2节回顾探索性数据分析和模式发现方面的相关工作。第 3 节介绍形式化建模背景。第4节阐述 ANALYZE 运算符的设计与语义。第5节提出我们的多查询优化策略。第6节报告我们的实验评估。第7节通过讨论研究发现与未来方向作结。

2 相关工作

自动化探索性数据分析。探索性数据分析（EDA） [MS20] 允许数据分析师与数据集管理工具进行交互，以获取亮点。EDA 的目标是生成亮点，从而发现数据子空间（可能是查询结果）中有趣、出乎意料且重要的事实，并使用数据叙述方法 [Sar99, SAM98, Sar00, SS01, IPC15, THY+17, EMS19, DHX+19, WSZ+20, MDHZ21, PAB+21, SXS+21, BRH+22, AMP23, SCC+23, LYZ+23, MDW+23, Ame24, XWJ24, LMS+25]呈现这些事实。多年来，人们提出了若干算子，用于补充基础算子。*DIFF* 算子 [Sar99]返回一组元组，这些元组能够最有效地描述输入的立方体两个单元格之间的值差异。同一作者还描述了一种对用户探索过程进行分析的方法，并利用最大熵原理推荐在后续查询中立方体哪些尚未访问的部分可能最令人惊讶 [Sar00]。最后，*RELAX* 算子允许验证在某一细节层级观察到的模式是否也存在于更粗的细节层级中 [SS01]。

Cinecubes 方法[GV13, GVM15] 中也提出了其他算子。这项工作的目标是：给定一个原始联机分析处理查询作为输入，实现自动化报告。为此，提出了两个算子

（表示为作用），即 (a)置于上下文，也就是说，将原始查询的结果与针对相似的同级值的查询结果进行比较；以及 (b)提供详细信息，即对原始查询的分组器执行下钻。

意图模型。意图模型 [VM18, VMR19]设想了这样一种商业智能工具：用户可对数据应用意图算子，从而简化数据分析操作中的查询步骤。用户提出“分析”“评估”“预测”等高层需求，这些需求需通过辅助查询、机器学习模型、亮点和数据故事来满足，包括：(a) 收集相关数据的辅助查询；(b) 模型（在机器学习语境下，指从收集的数据中提取模式）；(c) 亮点，即数据和模型中有趣的部分，并根据数据趣味性进行排序；(d) 数据故事，利用数据可视化和叙事方法展示发现。尽管意图模型的总体框架已在[VM18, VMR19], 中作出规定，但算子的具体实现仍是一个尚待解决的问题（目前，我们正针对 ANALYZE 运算符为此作出贡献）。[FGM+23, FGM+21] 探讨了评估算子，用于将查询结果与指定的预期性能“基准”进行比较。 [FMPR22]介绍了描述算子，它会生成带有模型组件（例如聚类、离群值）标注的数据立方体。 [FRM24] 提出了一种 EXPLAIN 算子，利用统计模型解释某个度量为何会相对于其他度量取特定值。与本文最相似的工作，也是意图模型的起源，是 Cinecubes 方法 [GV13, GVM15], ；该方法将辅助下钻查询和兄弟查询的结果制作为 PowerPoint 演示文稿，从而生成数据故事。

多查询优化 (MQO)。[Sel88]确立了基础性证据，表明相比于独立执行查询，同时处理多个查询可以大幅降低成本。Sellis 提出了在各种可能的组合执行计划中搜索的算法，权衡物化共享子计划的成本与复用计算带来的收益，并制定启发式方法，以控制现实工作负载下的评估成本。作为对上述工作的后续研究，[SG90] 首次对 MQO 进行了正式阐述，将其定义为：通过识别并利用公共子表达式、共享连接路径和可复用的中间结果，为一组并发查询生成最优执行策略。研究还引入了用于评估共享计划的成本模型扩展，使优化器能够推理物化与复用。 [RS09] 讨论了对 Volcano 优化器进行的全面实验验证。 [HRK+09、LKDL12]提供了额外的重写技术。[KS15] 提出了近似优化技术。 [RS09] 讨论了全面的实验验证，证明通过所提出的基于 Volcano 的启发式算法，可以在优化开销可接受的情况下显著提升性能。基于 Volcano 的算法旨在提高 MQO 的计算效率，并使其能够扩展到复杂查询和成本模型。该研究探讨了将 MQO 分解为易于处理的子问题所需的算法原语（例如，有界成本搜索、按相似性对查询进行分组），并将这些原语整合到优化器中，使其相较于先前的穷举方法能够扩展到更大的工作负载。 [HRK+09]

提出了一种基于规则的方法，利用声明式转换规则来检测并重构查询之间的共享计算。这种基于规则的设计会合并等价的子表达式、提取公共算子，并在成本允许时引入共享的物化结果。 [WC13] 报告称，其方法在 MapReduce 框架中取得了优于最先进技术的性能。作者制定了用于识别多个 MapReduce 作业之间公共计算的方法。在此背景下，MapReduce 作业会被重写或重新调度，使共享的 map 或 reduce 阶段只执行一次，并在各作业之间复用其输出。 [LKDL12] 提出了识别公共子图模式（例如连接模体）的技术，并利用这些模式构建可同时服务于多个 SPARQL 查询的统一执行计划。作者提出了结构索引和图模式聚类方法，以减少多查询计划枚举中的组合爆炸。除了性能提升， [KS15] 还通过为 MQO 算法提供首个近似因子保证，推进了其理论基础。作者提出了成本感知分组和计划构建技术，利用查询工作负载的结构特征，无需穷举枚举即可发现有益的共享子表达式。

与最先进技术的比较。本工作的主要贡献在于正式引入了一种新型算子及其优化技术。尽管在概念上相关，但本文的工作并不涉及多查询优化领域的核心：我们的多查询优化算法不探索替代的查询执行计划，而是利用正确性的理论保证。与最相似的研究方向 [GV13、GVM15], 相比，我们改变了查询语义，以解决效率问题（因此我们面对的是一个新问题），为算子提供形式语义；更重要的是，我们引入了多查询优化策略，以加快查询处理。与相关文献中的其他工作相比，据我们所知，还没有其他工作针对一个形式化的单一算子提出多查询优化策略；该算子通过整合各种查询，为数据提供 360^o 视图，并恰恰因此拥有优化机会。

3 预备知识与形式化背景

在我们的论述中，我们采用 [Vas22]（在[Vas23]中也使用）的形式化模型，用于定义多维空间、数据立方体和立方体查询。

3.1 多维空间

多维空间。数据是在多维空间的上下文中定义的。多维空间包含一个有限的维度集合。维度为事实度量提供上下文，并按照维度层级进行组织；这些抽象层级有助于以不同粒度观察数据。例如，维度 $Time$

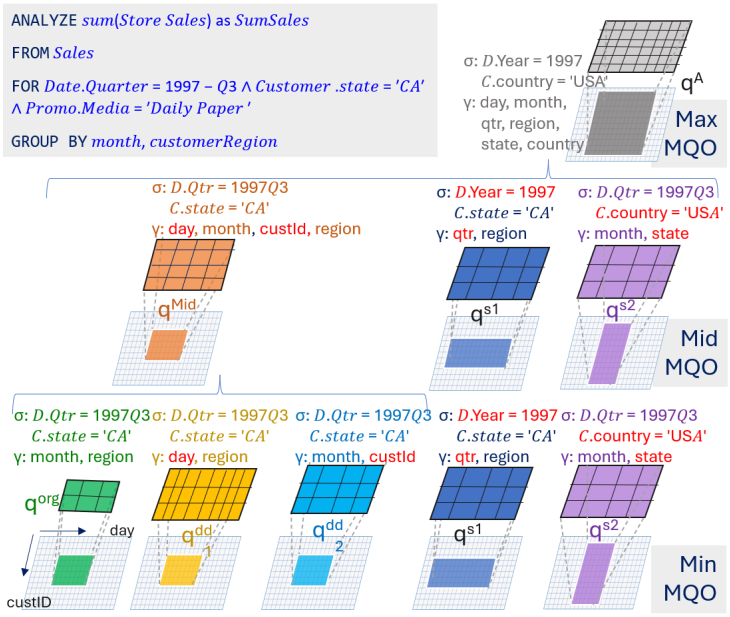


图1：每种算法的辅助查询

基于维度层级 $Day, Month, Year, All$ 进行组织。一个维度层级 L 包含一个名称和一个有限的值集合 $dom(L)$，作为其定义域。按照传统的联机分析处理术语，属于各层级定义域的值称为维度成员，或简称成员（例如，巴黎、罗马、雅典是层级 $City$ 定义域的成员，因此也是维度 $Geography$ 的成员）。一个维度 是一个有限层级集合上的非严格偏序，必须包含：(a) 粗粒度最低的最详细层级；(b) 一个上界，称为 ALL，其唯一值为“全部”。我们用 $\preceq$ 表示维度的偏序，即 $D.L_{low} \preceq D.L_{high}$ 表示在维度 D 的上下文中，$D.L_{low}$ 的粗粒度低于 $D.L_{high}$ ——例如，$Geo.City \preceq Geo.Country$。为简化表示，除非另有明确说明，以下假定层级构成全序，这意味着层级之间存在一个线性序列，从较低层级开始，以全部结束，并按优先关系形成一条线性链。

我们可以通过祖先函数 $anc_{L_l}^{L_h}()$，将粗粒度较低层级的成员映射为粗粒度较高层级的值。给定层级 L_l 中的一个成员作为参数，例如 v_l，函数 $anc_{L_l}^{L_h}()$ 会返回对应的祖先值，例如层级 L_h 中的 v_h，即 $v_h = anc_{L_l}^{L_h}(v_l)$。祖先函数的逆不是一个函数，而是将高层级值映射到粗粒度较低层级的一组后代值 （例如， $Continent$ 欧洲

会被映射到层级 $City$ 中所有欧洲城市构成的集合），记作 $desc_{L_h}^{L^l}$ ()。例如 $Europe = anc_{City}^{Continent}$(*Athens*)。有关更多约束和说明，请参阅[Vas22]。某个维度所有层级的定义域的并集也将宽泛地称为该维度的定义域，$dom(D)$。多维空间由各维度定义域的笛卡尔积生成，因此，多维空间中的每个点 c 都带有坐标 $c(c_1, \ldots, c_n)$，且每个维度恰有一个值。

数据立方体。事实以数据立方体的形式组织。数据立方体 C 相对于若干维度定义，这些维度固定在特定层级上，并且还包含若干度量，用于承载其事实中可度量的方面。因此，数据立方体的模式是一组属性，其中包括一组维度层级（来自不同维度）和一组度量，这些度量包含存储在数据立方体中的数据的事实度量。

形式上，数据立方体 $schema(C)$ 的模式是一个元组，例如 $[D_1.L_1, ..., D_n.L_n, , M_1, ..., M_m]$，或简写为 $[L_1, ..., L_n, M_1, ..., M_m]$。维度层级的组合（每个层级都来自不同维度）充当主键和度量的上下文，而一组度量则充当（聚合）度量的占位符。

如果数据立方体模式中的所有维度层级都是其所属维度的最低层级，则该数据立方体称为详细立方体，通常用记号C^0表示，其模式为 $[D_1.L_1^0, ..., D_n.L_n^0, M_1^0, ..., M_m^0]$。查询 q的结果是一组单元格，记作 $q.cells$。模式为$[D_1.L_1, \ldots, D_n.L_n, M_1, \ldots, M_m]$的数据立方体 C 中的每条记录（也称为单元格）都是一个元组 $c = [l_1, \ldots, l_n, m_1, \ldots\ldots, m_m]$，满足 $l_i \in dom(D_i.L_i)$ 和 $m_j \in dom(M_j)$。向量$[l_1, \ldots, l_n]$表示单元格的坐标。等价地，$cell$ 可以视为数据立方体各维度所构成的多维空间中的一个点，并附有或包含一组度量。数据立方体包含有限个单元格，其扩展即由这些单元格构成。

总的来说，立方体数据库包含层次结构和数据立方体这两类彼此相关但各自独立的一等公民。立方体根据层次结构中的层级来定义，因此，它在由这些层次结构构成的多维层次空间中的位置由这些层级决定。

3.2 立方体查询

查询。立方体查询本身也是一个立方体，其定义包括：(a) 所基于的详细立方体，(b) 用于筛选出符合进一步处理条件的事实的选择条件，(c) 决定结果粗粒度的分组层级，以及 (d) 对立方体的部分或全部度量进行的聚合，该聚合与分组层级一同出现在最终结果中。

$$q = < C^0, \phi, [L_1, ..., L_n, M_1, ..., M_m], [agg_1(M_1^0), ..., agg_m(M_m^0)] >$$

我们假设选择条件是由原子筛选条件合取而成，形式为 $L = value$，或一般形式 $L \in \{v_1, \ldots, v_k\}$。

此类选择条件最终可以通过对 中各原子的详细等价项取合取，转换为其在详细层级上的等价选择条件。

ϕ 中各原子的详细等价项。具体而言，假设有一个原子 $L \in \{v_1, \ldots, v_k\}$，那么 $L^0 \in \{desc_L^{L^0}(v_1) \cup ... \cup desc_L^{L^0}(v_k)\}$，最终得到表达式 $L^0 \in \{v'_1, ..., v'_{k'}\}$，它是其详细等价项，称为详细代理。推导 ϕ^0 的原因是，ϕ^0 作为最详细层级上相应原子筛选条件的合取，可直接应用于 C^0，并且在粒度达到最详细层级时，产生的多维空间子集与 ϕ 完全相同。例如，假设 $Year \in \{2018, 2019\}$，其详细代理为 $Day \in \{2018/01/01, \ldots, 2019/12/31\}$。对于未被任何原子显式筛选的维度 D，可以等效地假设一个形式为 $D.ALL = all$ 的筛选条件。

我们还假设聚合函数 agg_i 包括诸如$\{sum, max, min, count, ...\}$之类的聚合函数，并具有各自众所周知的语义。

立方体查询语义。查询的语义如下：(i) 应用 ϕ^0，即针对 C^0 上的选择条件的详细等价项，并生成详细立方体的一个子集，例如 q^0，称为查询的详细区域；(ii) 将每个维度成员映射到分组层级所指定层级上的祖先值，并将坐标相同的元组归入同一个同坐标组；(iii) 对每个同坐标组，将聚合函数应用于其中各单元格的度量，从而为每个聚合度量生成一个值。

立方体查询本身也是一个立方体，其模式为 $[L_1, ..., L_n, M_1, ..., M_m]$，扩展则由查询结果的单元格（记为 $q.cells$）构成。

约束。在本文的背景下，为了表述清晰，我们采用了若干假设。首先，我们讨论的立方体查询只涉及一个度量。其次，我们假设结果严格包含两个聚合器层级。第三，对于所有原子筛选条件，我们假设：如果带有筛选条件的维度同时也是分组器，那么该原子筛选条件所处的层级不低于分组器的层级。

- 我们研究仅涉及一个度量的立方体查询
- 我们假设结果严格包含两个聚合器层级
- 对于所有原子筛选条件，我们假设：如果带有筛选条件的维度同时也是分组器，那么该原子筛选条件所表示的层级必须大于或等于分组器的层级。我们用 $D.L^\sigma$ 表示原子筛选条件的层级（$D.L^\sigma = v$），用 $D.L^\gamma$ 表示相应的分组器，并要求如果查询定义中存在这样一个有序对，则 $D.L^\gamma \preceq D.L^\sigma$

上述约束的理由如下：

- 为避免符号过于繁杂，为清晰起见，我们选择只处理一个度量。将我们的结果推广到多个度量，只需按维度层级连接各个单度量立方体查询的结果，过程十分直接。
- 我们也选择只使用恰好 2 个分组器层级。这是一项传统且合理的约束，源于呈现方面的限制：结果必须显示在二维屏幕上，无论是以表格形式还是图表形式。

- 对过滤器和分组器之间关系的约束，涉及 [Vas22] 的完美可滚动性概念。例如，假设我们按 $city$ 对结果进行分组，那么如果对 $Geography$ 维度应用筛选条件，就应涉及特定的 $city$，或 $country$ （其度量按城市分组），或 $continent$。如果我们筛选某个特定的 $neighborhood$，然后要求按城市分组，虽然 (a) 可以得到结果，但 (b) 其语义至少并不明确，而且无论如何 (c) 都会违反完美可滚动性要求。因此，我们认为这项约束合理且实用。

接下来，我们将采用以下查询表达式：

$$q = < C^0, \phi, [L_\alpha, L_\beta, M], agg(M^0) >$$

4 ANALYZE 运算符

本节将从语义、语法和执行策略方面介绍 ANALYZE 运算符。

4.1 语法与语义

假设采用第 3 节中已经讨论过的设置，其中在层次维度列表上定义了详细立方体 C^0。ANALYZE 运算符的语法如下：

ANALYZE	$agg(M^0)$ as M FROM C^0
FOR	ϕ
GROUP BY	L_α, L_β
AS	$queryName$

其中，参与定义的变量具有已在第 3 节中介绍过的显然语义。

为了补充这一类似 SQL 的定义，我们还将采用该算子的代数表示，即 $analyze(C^0, \phi, [D_\alpha.L_\alpha, D_\beta.L_\beta, M], agg(M^0))$，或等价地表示为：

$$analyzeOp = \langle C^0, \phi, [D_\alpha.L_\alpha, D_\beta.L_\beta, M], agg(M^0) \rangle$$

示例。为阐释我们的讨论，我们将采用广为人知的 Foodmart 模式 [Hyd25] 的立方体化版本。Foodmart 包含一个名为 Sales 的立方体，具有 3 个度量，并由 5 个维度构成上下文，分别是 Product、日期、促销、客户和商店，每个维度都有自己的层级。所有维度均包含层级 ALL，作为粗粒度最高的层级。第 1 节中展示的查询将用作我们的参考示例。

语义。该算子的语义涉及引入三组独立的中间立方体查询，我们称之为辅助查询，每组都会生成一组不同的立方体查询结果。假设给定一个 ANALYZE 查询 AQ，$AQ.queries$ 的查询集是一个查询三元组

集 $< Q^{org},\ Q^{sib},\ Q^{dd} >$，而 AQ、 $AQ.result$ 的结果是一个三元组，包含三个查询集中各自查询的结果。

该算子生成的 3 个查询集如下：

- (a) Q^{org} 是一个单元素集合，其中恰好包含一个查询 q^{org}，我们将其称为原始查询，该查询根据算子的变量定义计算 M^0 的聚合值。

- (b) Q^{sib} 是一个兄弟查询集合，旨在比较算子定义中的关键值与其同级值的行为。

- (c) Q^{dd} 是一个下钻查询集合，旨在针对原始查询生成的值，为分析师提供更详细的数据。

图1展示了参考示例中的查询，以及后续多查询优化算法将生成的辅助查询。接下来，我们将介绍这些辅助协调器查询各自的语义和原理。

4.1.1 兄弟查询

关于兄弟查询的关键词是上下文：兄弟查询旨在将（a）与原始查询相关的数据空间的不同切片与（b）其同类进行比较，以便将其置于上下文中。通过将数据切片与其同类进行比较，数据切片实际上由分析师加以评估，从而被情境化。

原始构想。直观地说，给定一个原始查询，兄弟查询与其的区别如下：它不包含原始查询的原子选择公式 $L_i = v_i$ ，而包含形式为 $L_i \in children($ $parent(v_i))$的公式。

形式上，假设一个原始查询 q^{org}

$$q^{org} = \langle\ C^0,\ \phi_\alpha \wedge \phi_\beta \wedge \phi_\square,\ [D_\alpha.L^\gamma_\alpha,\ D_\beta.L^\gamma_\beta, M],\ agg(M^0)\ \rangle$$

并具有以下约束：

- ϕ_α 一个可选的选择原子，形式为： $D_\alpha.L^\sigma_\alpha = v_\alpha$, $L^\gamma_\alpha \preceq L^\sigma_\alpha$
- ϕ_β 一个可选的选择原子，形式为： $D_\beta.L^\sigma_\beta = v_\beta$, $L^\gamma_\beta \preceq L^\sigma_\beta$
- $\phi_\square$ 针对分组器以外维度的可选原子合取 D_α， D_β，其全部原子 ϕ_i 的形式为，ϕ_i： $L_i = v_i$

请注意，对于每个分组维度，我们区分分组级别（例如， L^γ_β）与筛选层级，例如， L^σ_β。在我们的示例中，我们按 L^γ_β： $state$分组，而我们的 L^σ_β 级别为 $country$ （$country = 'USA'$），高于分组器所在级别。

在最简单的情况下，关于维度（例如 D_α）的 q^{org} 兄弟查询将是

$$q^{s^\alpha} = \langle\, C^0,\, \phi^\star_\alpha \wedge \phi_\beta \wedge \phi_\square,\, [D_\alpha.L^\gamma_\alpha,\, D_\beta.L^\gamma_\beta, M],\, agg(M^0)\,\rangle$$

其中 $\phi^\star_\alpha$ 是形式为 $D_\alpha.L^\sigma_\alpha = v^\star_\alpha$ 的原子，并满足约束：$anc^{L^\sigma_{\alpha+1}}_{L^\sigma_\alpha}(v_\alpha) = anc^{L^\sigma_{\alpha+1}}_{L^\sigma_\alpha}(v^\star_\alpha)$。

换句话说，我们将恰好一个原子的选择值替换为一个同级值（通过约束表明它们的母值——高一个层级——相同）。在我们的示例中，如果将约束 $quarter = 1997-Q3$替换为 $year = 1997-Q4$，并保持查询的其余部分不变，就会为日期维度创建一个兄弟查询。有关一般情况的详细讨论，请参阅 [GVM15] 。

兄弟计算的一种便捷变体的定义。这里一个明显的问题是，如果选择原子中使用的某个值有太多同级值，即使我们只为这个维度创建兄弟查询，也会产生大量查询。如果将所有选择原子都考虑在内，这个问题还会进一步扩大。为避免这种复杂性，我们通过以下方式简化兄弟生成：(i) 仅考虑原子 ϕ_α 和 ϕ_β 的兄弟查询；(ii) 略微调整聚合层级，将所有兄弟切片合并为一个。

给定原始查询 q^{org}

$$q^{org} = \langle\, C^0,\, \phi_\alpha \wedge \phi_\beta \wedge \phi_\square,\, [D_\alpha.L^\gamma_\alpha,\, D_\beta.L^\gamma_\beta, M],\, agg(M^0)\,\rangle$$

我们针对每个分组器生成一个兄弟查询，如下所示：

$$q^{s^\alpha} = \langle\, C^0,\, \phi^\star_\alpha \wedge \phi_b \wedge \phi_\square,\, [D_\alpha.L^\sigma_\alpha,\, D_\beta.L^\gamma_\beta, M],\, agg(M^0)\,\rangle,\, \phi^\star_\alpha\colon\, L^\sigma_{\alpha+1} = anc^{L^\sigma_{\alpha+1}}_{L^\sigma_\alpha}(v_\alpha)$$

$$q^{s^\beta} = \langle\, C^0,\, \phi_\alpha \wedge \phi^\star_\beta \wedge \phi_\square,\, [D_\alpha.L^\gamma_\alpha,\, D_\beta.L^\sigma_\beta, M],\, agg(M^0)\,\rangle,\, \phi^\star_\beta\colon\, L^\sigma_{\beta+1} = anc^{L^\sigma_{\beta+1}}_{L^\sigma_\beta}(v_\beta)$$

_直观来说，我们对原始查询进行以下两项修改：

•我们不再询问正在调节的维度在筛选层级上的值，而是询问其母节点值。因此，我们不再使用 a$state = CA$ 筛选条件，而是引入一个筛选条件 $mama(state) = mama(CA)$，即 $country = USA$。

• 我们按照原来的筛选层级进行聚合。在本例中，既然我们已选择整个 USA 作为要处理的子空间，就按照原来的选择器层级 $state$进行分组，从而将州作为结果的分组器值，并有效地将 USA中的所有兄弟州相互比较（包括作为参照项的州 CA）。

在我们的示例中：

$q^{org} = \langle\ Sales,\ \ Date.Quarter\ =\ 1997Q3 \wedge Customer.State\ =\ 'CA' \wedge Promo.media\ =\ 'Daily\ Paper',\ \ [Date.Month, Customer.Region, MORG],\ sum\ (StoreSales)\ \ \rangle q^{s^{Date}} = \langle\ Sales,\ Date.Year\ =\ '1997'\ Customer.State\ =\ 'CA' \wedge Promo.media\ =\ 'Daily\ Paper',\ \ [Date.Quarter, Customer.Region, MSDATE],\ \ sum\ (StoreSales)\ \ \rangle q^{s^{Cust}} = \langle\ Sales,\ Date.Quarter\ =\ 1997Q3 \wedge Customer.Country\ =\ 'USA' \wedge Promo.media\ =\ 'Daily\ Paper',\ \ [Date.Month,Customer.State,\ MSCUST],\ sum\ (StoreSales)\ \ \rangle$

4.1.2 下钻查询

除了将原始查询的结果与同类结果进行对比以提供背景信息外，我们还可以通过深入探查生成结果中的维度层次结构，为显示的数据提供更多细节。理想情况下，这需要深入探查原始查询结果中的每个单元格——参见 [GVM15]，了解传统做法。然而，这会产生数量极大的待执行查询。我们采用一种简单的方法来解决这个问题：深入探查每个聚合器层级，在其层次结构中向下移动一个层级。因此，我们生成两个下钻查询，分别沿一个分组维度向下探查一个层级，以获取详细信息。

给定原始查询 q^{org}

$$q^{org} = \langle\ C^0,\ \phi,\ [D_\alpha.L^\gamma_\alpha,\ D_\beta.L^\gamma_\beta, M],\ agg(M^0)\ \rangle$$

我们会生成两个下钻查询，分别对应其中一个分组器，如下所示：

$$q^{dd^\alpha} = \langle\ C^0,\ \phi,\ [D_\alpha.L^\gamma_{\alpha-1},\ D_\beta.L^\gamma_\beta, M^{dd^\alpha}],\ agg(M^0)\ \rangle$$

$$q^{dd^\beta} = \langle\ C^0,\ \phi,\ [D_\alpha.L^\gamma_\alpha,\ D_\beta.L^\gamma_{\beta-1}, M^{dd^\beta}],\ agg(M^0)\ \rangle$$

在我们的示例中：

$q^{org} = \langle\ Sales、\ \ Date.Quarter\ =\ 1997Q3 \wedge Customer.State\ =\ 'CA' \wedge Promo.media\ =\ 'Daily\ Paper'、\ [Date.Month, Customer.Region, MORG]、\ sum\ (StoreSales)\ \ \rangle$

$q^{s^{Date}} = \langle\ Sales,\ Date.Year\ =\ '1997'\ Customer.State\ =\ 'CA' \wedge Promo.media\ =\ 'Daily\ Paper',\ [Date.Quarter, Customer.Region, MSDATE],\ sum(StoreSales)\ \ \rangle$

$q^{s^{Cust}} = \langle\ Sales,\ Date.Quarter\ =\ 1997Q3 \wedge Customer.Country\ =\ 'USA' \wedge Promo.media\ =\ 'Daily\ Paper',\ [Date.Month, Customer.State,MSCUST],\ sum(StoreSales)\ \ \rangle$

$$q^{dd^{Date}} = \langle\ Sales,\ Date.Quarter\ =\ 1997Q3 \wedge Customer.State\ =\ 'CA' \wedge Promo.media\ =\ 'Daily\ Paper',\ [Date.Day,\ Customer.Region,\ MDDDate],sum(StoreSales)\ \rangle q^{dd^{Cust}} = \langle\ Sales,\ Date.Quarter\ =\ 1997Q3 \wedge Customer.State\ =\ 'CA' \wedge Promo.media\ =\ 'Daily\ Paper',\ [Date.Month,\ Customer.CustomerId,\ MDDCust],\ sum(StoreSales)\ \rangle$$

5 ANALYZE 查询的多查询优化

目前我们已经看到，ANALYZE 运算符实际上是对底层数据库执行 5 个辅助查询的组合。在本节中，我们将介绍一种方法：通过合并这些查询中的（一个子集），取代向数据库发出五个查询的需求；同时介绍相应的算法，通过发出单个查询来（a）以适当的聚合层级收集这 5 个查询所需的全部数据，以及（b）在内存中对这些数据进行后处理，从而正确生成这 5 个查询的结果。正如后文将要说明的，这种方式一方面成本高昂，因为多个查询的成本会累加查询开销；另一方面，这种开销也是可以避免的。

该方法旨在通过以下方式提升速度：(a) 避免查询开销；(b) 将对同一数据的多次请求减少为从数据库中获取一次（即使在朴素执行中，DBMS 内部缓存也会在一定程度上降低这一成本）；(c) 我们后处理的单元格与查询结果非常接近，因此需要后处理的信息量很少——由此，我们可以在主存中高效地完成这一工作。

5.1 理论背景

我们的理论构造基于立方体可用性定理 [Vas22，Vas23]，该定理指出，如果满足某些条件，就可以通过对已访问的数据应用另一个立方体查询进行过滤并重新聚合，从而得出立方体查询的结果。

为确保内容完整，我们在此重述该定理的表述，并请读者参阅相关论文，了解证明细节及其支撑概念。

定理 5.1（立方体可用性（[Vas23，Vas22]））。假设有以下两个查询：

$$q^n = \langle \mathbf{DS}^0,\ \phi^n,\ [L_1^n, \ldots, L_n^n, M_1, \ldots, M_m],\ [agg_1(M_1^0), \ldots, agg_m(M_m^0)] \rangle$$

and

$q^b = \langle \mathbf{DS}^0,\ \phi^b,\ [L_1^b, \ldots, L_n^b, M_1, \ldots, M_m],\ [agg_1(M_1^0) \ldots, agg_m(M_m^0)] \rangle$

如果满足以下条件，则查询 q^b 可用于计算，或简称为可用于查询 q^n:

i. 两个查询具有完全相同的底层详细立方体 **DS**,

ii. 两个查询的模式中具有完全相同的维度和相同的聚合度量 $agg_i(M_i^0)$、$i \in 1..\ m$ （这意味着其度量之间存在一一映射），且所有 agg_i 都属于一组已知的可分配函数。

iii. 两个查询的选择条件中每个维度恰好有一个原子，其形式为 $D.L \in \{v_1, \ldots, v_k\}$，且选择条件是此类原子的合取。

iv. 两个查询的模式相对于其选择条件都可完美上卷，这意味着分组器层级相对于其维度中对应的原子都可完美上卷,

v. 查询 q^n 的所有模式层级都是 q^b中对应层级的祖先（即相同或更高），也就是说，对于所有维度 D，都有 $D.L^b \preceq D.L^n$，并且,

vi. 对于 ϕ^n中的每个原子，例如 α^n，如果 (i) 我们得到 $\alpha^{n@L^b}$ （即前一个查询 q^b、 L^b的对应模式层级上的详细等价项），并将其简称为 $\alpha^{n@b}$，且 (ii) 计算其特征 $\alpha^{n@b^+}$，那么 (iii) 该特征是 q^b 处对应维度的分组器定义域的子集（其中涉及对应的原子 α^b 和分组级别 L^b），也就是说，$\alpha^{n@b^+} \subseteq gdom(\alpha^b,\ L^b)$。

在我们的情形中，我们为 ANALYZE 运算符引入辅助的全涵盖查询 q^A 。全涵盖查询 q^A 具有以下特征：（0）相同的基本立方体和度量聚合，（1）分组器涉及足够低、从而能够回答 ANALYZE 运算符任何内部查询的最高聚合层级，以及（2）尽可能宽泛的选择条件，以涵盖回答任何内部查询所需的全部详细元组。在本节的其余部分，我们将首先证明全涵盖查询 q^A 能够回答所有内部查询，其次，我们将用两个实际计算答案的算法来配合这一可行性结果。

定理 5.2 （多查询可用性）。 假设查询$analyze = \langle\ C^0, \phi_\alpha \wedge \phi_\beta \wedge \phi_\square, [D_\alpha.L_\alpha^\gamma,\ D_\beta.L_\beta^\gamma, M], agg(M^0)\ \rangle$按照 ANALYZE 运算符的定义生成五个内部查询。以下全涵盖查询 q^A 可以回答 ANALYZE 运算符的所有内部查询。

q^A=⟨C^0、 $\phi^\star_\alpha \wedge \phi^\star_\beta \wedge \phi_\square$、 $[L^\gamma_{\alpha-1}, L^\gamma_{\beta-1}, L^\gamma_\alpha, L^\gamma_\beta, L^\sigma_\alpha, L^\sigma_\beta, M^A]$、 $agg(M^0)$⟩,
其中 $\phi^\star_\alpha$: $L^\sigma_{\alpha+1} = anc^{L^\sigma_{\alpha+1}}_{L^\sigma_\alpha}(v_\alpha)$, 以及 $\phi^\star_\beta$: $L^\sigma_{\beta+1} = anc^{L^\sigma_{\beta+1}}_{L^\sigma_\beta}(v_\beta)$

证明。 立方体可用性定理的部分要求由该算子和全涵盖查询的构造所提供。我们请读者参阅表1，以清楚地说明为何下列各项成立。具体而言：

i. 所有查询都针对同一个详细立方体。ii. 所有查询具有相同的维度、相同的度量以及相同的可分配聚合函数。iii. 每个查询在每个维度上都有一个形式为 $D.L = v$ ($true$ 的原子，或者等价地说，隐含着 $D.ALL = all$ 个原子。事实上，所有查询恰好在相同维度上具有非平凡原子；更进一步，它们可能仅在分组维度的原子上有所不同。iv. 根据构造，所有查询的选择器层级都高于分组器层级。我们已经假定，对于所有维度 D，若其分组级别为 $D.L^\gamma$，且选择条件中 D、 $D.L^\gamma \preceq D.L^\phi$ 的原子所涉及的层级为 $D.L^\phi$，则选择条件是在高于分组的层级上定义的。因此，根据构造，$D.L^\gamma_{i-1} \preceq D.L^\gamma_i \preceq D.L^\sigma_i$。v. 所有查询的分组级别都高于或等于涵盖所有查询的那个查询的分组级别（再次提醒，记住 $D.L^\gamma_i \preceq D.L^\sigma_i$）。vi. 除了分组维度的原子之外，所有查询中的选择条件都相同（也就是说，它们共享相同的 $\phi_\square$）。正如我们马上将要展示的，分组维度的原子也满足约束 (vi)。

因此，我们只需证明第六个约束 (iv)：一旦计算出全涵盖查询，就可以对其结果应用一个选择条件，从而得到与原始查询相同的结果。我们先证明，给定应用了 $\phi^\star_\alpha$ 后计算出的全涵盖查询结果，可以通过应用 ϕ_α 得到完全相同的结果（ β 的证明完全相同）。

我们来总结一下关于维度为 D_α 的选择原子的情况。

- “前一个”查询是范围最广的查询， q^A，它具有分组层级 $D_\alpha.L^\gamma_{\alpha-1}$ 和选择条件 $\phi^\star_\alpha$: $L^\sigma_{\alpha+1} = anc^{L^\sigma_{\alpha+1}}_{L^\sigma_\alpha}(v_\alpha)$。因此，实际生成的 q^A 值分组器定义域为 D_α ： v^g: 满足 $anc^{L^\sigma_{\alpha+1}}_{L^\gamma_{\alpha-1}}(v^g) = anc^{L^\sigma_{\alpha+1}}_{L^\sigma_\alpha}(v_\alpha)$，也就是说， v_α的母节点（v_α的兄弟查询）在层级 $D_\alpha.L^\gamma_{\alpha-1}$上的子节点。

	C^0	ϕ: $\phi_\alpha \wedge \phi_\beta \wedge \phi_\square$			模式			agg
	$[i]$	($[iii][iv][vi]$)			($[ii][iv][v]$)			$[ii]$
q^A	C^0	$\phi^\star_\alpha$	$\phi^\star_\beta$	$\phi_\square$	$D_\alpha.L^\gamma_{\alpha-1}$	$D_\beta.L^\gamma_{\beta-1}$	M^A	$agg(M^0)$
q^{org}	C^0	ϕ_α	ϕ_β	$\phi_\square$	$D_\alpha.L^\gamma_\alpha$	$D_\beta.L^\gamma_\beta$	M^{org}	$agg(M^0)$
q^{dd^α}	C^0	ϕ_α	ϕ_β	$\phi_\square$	$D_\alpha.L^\gamma_{\alpha-1}$	$D_\beta.L^\gamma_\beta$	M^{dda}	$agg(M^0)$
q^{dd^β}	C^0	ϕ_α	ϕ_β	$\phi_\square$	$D_\alpha.L^\gamma_\alpha$	$D_\beta.L^\gamma_{\beta-1}$	M^{ddb}	$agg(M^0)$
q^{s^α}	C^0	$\phi^\star_\alpha$	ϕ_β	$\phi_\square$	$D_\alpha.L^\sigma_\alpha$	$D_\beta.L^\gamma_\beta$	M^{ssa}	$agg(M^0)$
q^{s^β}	C^0	ϕ_α	$\phi^\star_\beta$	$\phi_\square$	$D_\alpha.L^\gamma_\alpha$	$D_\beta.L^\sigma_\beta$	M^{ssb}	$agg(M^0)$

表1：全涵盖查询的多查询可用性证明表。表头中的数字对应立方体可用性定理中相应要求的 ID。

查询	D_α: 日期	D_β: 客户	$Grouper_\alpha$	$Grouper_\beta$
q^A	$Year = '1997'$	$Country = 'USA'$	$[Day$	$Customer]$
q^{org}	$Quarter = 1997Q3$	$State = 'CA'$	$[Month$	$Region]$
$q^{dd^{Date}}$	$Quarter = 1997Q3$	$State = 'CA'$	$[Day$	$Region]$
$q^{dd^{Cust}}$	$Quarter = 1997Q3$	$State = 'CA'$	$[Month$	$Customer]$
$q^{s^{Date}}$	$Year = '1997'$	$State = 'CA'$	$[Quarter$	$Region]$
$q^{s^{Cust}}$	$Quarter = 1997Q3$	$Country = 'USA'$	$[Month$	$State]$

表 2：为理论解释而重新审视的参照示例

- 关于 D_α，所有查询都包含一个原子 ϕ_α: $L^\sigma_\alpha = v_\alpha$，或一个原子$\phi^\star_\alpha$: $L^\sigma_{\alpha+1} = anc^{L^\sigma_{\alpha+1}}_{L^\sigma_\alpha}(v_\alpha)$。
- 对于包含原子 $\phi^\star_\alpha$ 的“新”查询，无需做任何处理：这个涵盖所有情况的查询已将定义域限制为恰好所需的值。

- 其余查询必须将 q^A 返回值列表调整为它们实际需要的值，也就是满足 ϕ_α 的值。如果是这种情况，就必须对 q^A 的结果应用适当的选择条件，以便只选出必要的值。将 ϕ_α 原子转换到 q^A 模式级别， $D_\alpha.L^\gamma_{\alpha-1}$为: $D_\alpha.L^\gamma_{\alpha-1} \in desc^{L^\gamma_{\alpha-1}}_{L^\sigma_\alpha}(v_\alpha)$，它会产生 v_α 的子节点，而这些子节点显然是 q^A 的分组器定义域的子集。因此，如果我们对 q^A 的结果应用此选择谓词，就能将 D_α 值集限制为恰好所需的值。

对 ϕ_β 和 $\phi^\star_\beta$ 应用相同的过程。

这样，检查清单中的 (vi) 项便得到了证明，因此整个定理成立。

□

回到我们的示例，表 2 仅展示了查询定义中相关的部分。请注意，对于 $Date$ 维度， q^A 产生的值是 1997年中的天（分组级别为 Day）（选择原子为 $Year = 1997$），即 $1997/01/01$, $1997/01/02$ 等。现在，对于所有需要查询 1997年第三季度的查询，我们需要重新应用针对$Quarter = 1997Q3$的筛选条件，但分组级别要设为 q^A，即日！因此，应用于 q^A 结果的筛选条件将是 $Day \in desc^{Day}_{Quarter}(1997Q3)$。在较高层级进行筛选、在较低层级进行分组这一事实，确保了完美可滚动性 [Vas23, Vas22], ，从而使我们能够复用涵盖范围最广的查询结果。还请注意，由于 q^A的分组器层级低于或等于所有其他查询的分组器层级，我们还可以将 q^A 的值进一步分组到其余所有 ANALYZE 内部查询的更高层级。

5.2 最大多查询优化算法

MaxMQO算法 （算法 1）说明了如何通过单次访问底层数据库，计算出 ANALYZE 运算符的所有辅助查询。算法以原始查询的定义为输入，该定义足以确定全部辅助查询，并生成这些查询的结果作为输出。算法步骤如下。

步骤 1. 首先，算法构建五个映射表，每个映射表对应一个用于确定 ANALYZE 运算符的查询。每个映射表都将存储相应查询的结果。映射表的键是分组器值的组合（基本上就是每个结果单元格的坐标），值则是与这些坐标对应的聚合度量。

步骤 2. 其次，算法构建并执行全覆盖辅助查询 q^A ，以此作为计算所有其他查询结果的基础。

步骤 3. 计算出 q^A 的元组后，我们需要将它们分发到该算子的辅助查询中。我们逐个处理元组。所有元组最终都会进入兄弟查询，因为 q^A 的选择条件与兄弟查询的完全相同。请记住，兄弟查询的选择条件比原始查询和下钻查询（其选择条件与原始查询相同）更宽泛。不过，有些结果元组也属于原始查询和下钻查询，因此必须将它们放入相应的映射表。最终，哈希映射表中包含辅助查询的结果。

例如，假设用户发出以下$ANALYZE$ 查询：

ANALYZE 求和（门店销售额）来自销售额，按
州 = '加利福尼亚州' 和 季度 = '2025年第四季度'

算法 1： MaxMQO 算法

输入： 原始查询 $q^{org} = \langle\ C^0,\ \phi,\ [D_\alpha.L^\gamma_\alpha,\ D_\beta.L^\gamma_\beta, M],\ agg(M^0)\ \rangle$，
phi：$\phi_\alpha \wedge \phi_\beta \wedge \phi_\square$，$phi_i$：$D_i.L^\sigma_i = v_i$，$i$：$\alpha\ or\ \beta$

输出： q^{org}, q^{s^A}的更新后的结果 , q^{s^B}, q^{dd^A}, q^{dd^B}

开始

  设 $H^{org}, H^{s^A}, H^{s^B}, H^{dd^A}, H^{dd^B}$ 为如下形式的映射表
  H：$\langle\ dom(L^\gamma_1)\ dom(L^\gamma_2)\ \rangle \rightarrow dom(M^0)$；

  令 $aggF^\star = adapter(aggF)$;

  /* 构造并执行多查询优化查询 */

  令 $\phi^\star_\alpha$：$L^\sigma_{\alpha+1} = anc^{L^\sigma}_{L^\sigma_\alpha{}^{\alpha+1}}(v_\alpha)$ 和 $\phi^\star_\beta$：$L^\sigma_{\beta+1} = anc^{L^\sigma}_{L^\sigma_\beta{}^{\beta+1}}(v_\beta)$;

  令 $q^A = \langle\ C^0$、$\phi^\star_\alpha \wedge \phi^\star_\beta \wedge \phi_\square$、$[L^\gamma_{\alpha-1}, L^\gamma_{\beta-1}, L^\gamma_\alpha, L^\gamma_\beta, L^\sigma_\alpha, L^\sigma_\beta, M^A]$,
  $agg(M^0)\ \rangle$；

  $q^A.cells = q^A.execute()$;

  对结果集 $q^A.cells$ 中的元组 t **执行**

    假设 $t = \langle v^\gamma_{\alpha-1},\ v^\gamma_{\beta-1},\ v^\gamma_\alpha,\ v^\gamma_\beta,\ v^\sigma_\alpha,\ v^\sigma_\beta,\ m\rangle$;

    /* 如果 t 满足 $\phi_\alpha \wedge \phi_\beta$：更新组织与依赖关系 */

    如果 $\sigma_{\phi_\alpha \wedge \phi_\beta}(t)$ **则**

      $updateMap(H^{org}, \langle\langle v^\gamma_\alpha, v^\gamma_\beta\rangle, m\rangle, agg)$;

      $updateMap(H^{dd^A}, \langle\langle v^\gamma_{\alpha-1}, v^\gamma_\beta\rangle, m\rangle, agg)$;

      $updateMap(H^{dd^B}, \langle\langle v^\gamma_\alpha, v^\gamma_{\beta-1}\rangle, m\rangle, agg)$;

    end

    如果 $\sigma_{\phi_\beta}(t)$ **则**

      $updateMap(H^{s^A}\ \langle\langle v^\sigma_\alpha, v^\gamma_\beta\rangle, m\rangle, agg)$;

    end

    如果 $\sigma_{\phi_\alpha}(t)$**则**

      $updateMap(H^{s^B}\ \langle\langle v^\gamma_\alpha, v^\sigma_\beta\rangle, m\rangle, agg)$;

    end

  end

  $q^{org}.cells = H^{org}$; $q^{dd^A}.cells = H^{dd^A}$ $q^{dd^B}.cells = H^{dd^B}$;

  $q^{s^A}.cells = H^{s^A}$; $q^{s^B}.cells = H^{s^B}$;

  返回 $\langle q^{org}.cells,\ q^{s^A}.cells, q^{s^B}.cells, q^{dd^A}.cells, q^{dd^B}.cells\rangle$;

结束

函数 更新映射(H：map，$\langle\langle g_1,\ g_2\rangle,\ m\rangle$:
$Tuple\langle Pair\langle grouper, grouper\rangle, measure\rangle$, $aggF^\star$：$agg.\ Func.$)：

  如果 $H.containsKey(\langle g_1, g_2\rangle)$ **那么**

    $curValue = H.get(\langle g_1, g_2\rangle)$;

    $H.put(\langle g_1,\ g_2\rangle,\ aggF^\star(currValue,\ m))$;

  end

  else

    $H.put(\langle g_1, g_2\rangle, m)$;

  end

按州、季度分组

此查询被视为原始查询 q^{org}。步骤 1 为每个辅助查询的查询结果 H^{org}、H^{s^A}、H^{s^B}、H^{dd^A}、H^{dd^B} 构建五个映射表。在步骤 2 中，通过将 q^{org} 的选择条件替换为其父级及相应值，构建全涵盖查询 q^A 。此外，当前集合中还添加了四个分组器，其中包含 q^{org} 分组器的子级和父级。全涵盖查询如下：

ANALYZE 求和(门店销售额) FROM 销售额 FOR 国家 = '美国' AND 年 ='2025'分组 城市, 月份, 州, 季度, 国家, 年

然后，将此全涵盖查询转换为 SQL 并执行。结果包含所有辅助查询的结果元组。在步骤 3 中，遍历每个结果元组；如果它属于相应辅助查询的结果（q^{org}、q^{s^A}、q^{s^B}、q^{dd^A}、q^{dd^B}），则根据立方体可用性定理[Vas22, Vas23]，将其分配到步骤 1 中创建的结果桶 H^{org}、H^{s^A}、H^{s^B}、H^{dd^A}、H^{dd^B}。

还请注意元组是如何被整合到任一映射表中的。方法$updateMap()$ 实现了这一处理：（a）如果该元组的坐标在映射中尚不存在，我们必须将传入的元组插入该映射，作为其所在组的第一个元组。（b）否则，如果坐标已经存在，我们需要更新聚合度量。请注意，我们不能直接应用 agg 函数：虽然聚合函数可直接用于 min、max 和 sum的情形，但对于 $count$的情形，我们只需将计数器加一。为便于处理，我们引入了 适配器 聚合函数，通过相应地调整当前值与新值的实际组合来解决这一问题。

5.3 面向 Analyze 算子的中型多查询优化算法

最大多查询优化算法在底层数据库上使用单条查询来检索 $ANALYZE$ 查询的结果。然而，全覆盖辅助查询q^A 的选择条件会探查一个庞大的元组超集，其中既包含这五个查询各自的查询结果，也包含不属于任何查询结果、因而未被聚合的元组。这些多余的元组会增加性能延迟，因为最大多查询优化必须处理它们，以判断某个元组是否属于五个查询结果映射表之一。

为了缩小已探索数据空间的范围，我们可以减少q^A返回的冗余元组数量。注意，原始查询和下钻查询的选择条件相同。我们利用这一特性构造一个单一查询q^{MID} ，将原始查询和下钻查询的协调器查询合并，而不涉及兄弟查询。

中间多查询优化算法（算法2）会启动两个兄弟查询以及合并后的 q^{MID} 查询。由于后者涉及三个协调器查询，我们按原始层级和祖先层级对元组进行分组，以分发结果元组。这样，我们只需单次访问，就能获取对原始查询和下钻查询结果有贡献的元组。其余处理与最大多查询优化相同。与q^A相比，q^{MID}返回的详细元组更少，对这些元组应用的分组器也更少，因此结果大小更小。

算法2展示了中间多查询优化的实现。首先，该算法构建三个映射表：一个用于原始查询，另外两个分别用于确定$ANALYZE$ 算子的下钻查询。每个映射表都将存储相应查询的结果。映射表的键是分组器值的组合，值则是与这些坐标相关的聚合度量（即 <grouper1,grouper$_2$>->m）。生成的元组会通过 $updateMap()$ 放入相应的桶中，具体如第5.2节所述。

随后，分别构建并执行兄弟查询 q^{sA}、q^{sB}，以返回准确的兄弟结果元组。这样，我们通过应用仅涉及父值的选择条件，避免出现多余元组，而这些条件会导致对事实表的大范围探索。最后，将这三个查询的结果元组打包，生成 ANALYZE 查询结果。

例如，假设用户发出以下 $ANALYZE$ 查询：

ANALYZE 求和(门店销售额)FROM 销售额 FOR
州 = '加利福尼亚州' AND 季度 = '2025年第四季
度' 分组 州,季度

此查询被视为原始查询 q^{org}。中间多查询优化算法会为每个促进查询结果 H^{org}，H^{dd^A}，H^{dd^B} 构建三个映射表。随后，通过保留 q^{org} 的选择条件，并在当前集合中再添加两个分组器，构建合并查询 q^{MID}。合并查询的分组器包含 q^{org} 分组器的子层级。合并查询如下：

ANALYZE 求和(门店销售额) FROM 销售额
FOR 州 = '加利福尼亚州' AND 季度 = '2025
年第四季度' 分组 城市, 月份, 州, 季度

然后，根据第4节讨论的语法构建兄弟查询 q^{s^A}，q^{s^B}。接着，将合并查询 q^{MID} 和兄弟查询q^{s^A}，q^{s^B} 转换为 SQL 并执行。兄弟查询的结果包含准确的兄弟结果元组。合并查询包含原始查询和下钻查询的结果元组。与最大多查询优化算法中的描述类似，会逐一访问每个元组；如果该元组属于相应的促进查询（q^{org}，q^{dd^A}，q^{dd^B}），则将其分配到结果桶H^{org}，H^{dd^A}，H^{dd^B}中，这依据的是立方体可用性定理 [Vas22, Vas23]。

算法2： Mid 多查询优化算法

输入： 原始查询 $q^{org} = \langle C^0,\ \phi,\ [D_\alpha.L^\gamma_\alpha, D_\beta.L^\gamma_\beta, M],\ agg(M^0)\rangle$，
$\phi = \phi_\alpha \wedge \phi_\beta \wedge \phi_\square$，　$\phi_i:\ D_i.L^\sigma_i = v_i,\ \ i \in \{\alpha, \beta\}$

输出： q^{org} 的更新后的结果，q^{s^A}　q^{s^B}　q^{dd^A}　q^{dd^B}

开始

　令 H^{org}，H^{s^A}，H^{s^B}，H^{dd^A}，H^{dd^B} 为以下形式的映射表

　　H: $\langle dom(L^\gamma_1)\ dom(L^\gamma_2)\rangle \rightarrow dom(M^0)$;

　令 $aggF^\star = adapter(aggF)$;

　/* 构造并执行 Mid 多查询优化查询 */

　令 $\phi^*_\alpha : L^\sigma_{\alpha+1} = anc_{L^\sigma_{\alpha+1}L^\sigma_\alpha}(v_\alpha)$;

　令 $\phi^*_\beta:\ L^\sigma_{\beta+1} = anc_{L^\sigma_{\beta+1}L^\sigma_\beta}(v_\beta)$;

　Let
　$q^{org\&dd} = \langle C^0, \phi_\alpha \wedge \phi_\beta, [L^\gamma_{\alpha-1}, L^\gamma_{\beta-1}, L^\gamma_\alpha, L^\gamma_\beta, M^{org\&dd}], agg(M^0)\rangle$;

　令 $q^{s^A} = \langle C^0, \phi^*_\alpha \wedge \phi_\beta, [L^\gamma_{\alpha+1}, L^\gamma_\beta, M^{s^A}], agg(M^0)\rangle$;

　令 $q^{s^B} = \langle C^0, \phi_\alpha \wedge \phi^*_\beta, [L^\gamma_\alpha, L^\gamma_{\beta+1}\ M^{s^B}], agg(M^0)\rangle$;

　$q^{org\&dd}.cells \leftarrow q^{org\&dd}.execute()$;

　$q^{s^A}.cells \leftarrow q^{s^A}.execute()$;

　$q^{s^B}.cells \leftarrow q^{s^B}.execute()$;

　对每个 元组 $t \in q^{org\&dd}.cells$ **执行**

　　假设 $t = \langle v^\gamma_{\alpha-1}, v^\gamma_{\beta-1}, v^\gamma_\alpha, v^\gamma_\beta, m\rangle$;

　　如果 $\sigma_{\phi_\alpha \wedge \phi_\beta}(t)$ **则**

　　　$updateMap(H^{org}, \langle\langle v^\gamma_\alpha, v^\gamma_\beta\rangle, m\rangle, agg)$;

　　　$updateMap(H^{dd^A}, \langle\langle v^\gamma_{\alpha-1}, v^\gamma_\beta\rangle, m\rangle, agg)$;

　　　$updateMap(H^{dd^B}, \langle\langle v^\gamma_\alpha, v^\gamma_{\beta-1}\rangle, m\rangle, agg)$;

　　end

　end

　$q^{org}.cells \leftarrow H^{org}$;

　$q^{dd^A}.cells \leftarrow H^{dd^A}$, $q^{dd^B}.cells \leftarrow H^{dd^B}$;

　返回 $\langle q^{org}.cells, q^{s^A}.cells, q^{s^B}.cells, q^{dd^A}.cells, q^{dd^B}.cells\rangle$;

结束

函数 更新映射(H: $map, \langle\langle g_1, g_2\rangle, m\rangle$:
$Tuple\langle Pair\langle grouper, grouper\rangle, measure\rangle, aggF^\star$):

　如果 $H.containsKey\ (\langle g_1,\ g_2\rangle)$ **那么**

　　$curValue \leftarrow H.get(\langle g_1,\ g_2\rangle)$;

　　$H.put(\langle g_1, g_2\rangle, aggF^\star(curValue, m))$;

　end

　else

　　$H.put(\langle g_1, g_2\rangle, m)$;

　end

5.4 关于我们算法设计选择的说明

在介绍这些算法的实验评估之前，值得先简要说明指导其设计的基本选择。我们引入了一个算子，它最终会调用多个数据库查询（即使我们也提出了一种将它们合并为一个查询的算法）。此外，我们引入的优化（a）独立于数据库管理系统，因此与具体的数据库管理系统无关；（b）在查询重写的语法层面自动执行，而不是探索备选的多查询优化方案。

这里值得吸取的一条重要经验是，我们可以在直接数据库查询之上引入一个抽象层，使其更贴合分析师的思维方式（这与为联机分析处理引入上卷和下钻的做法颇为相似）。这些高层算子在内部可能涉及多个辅助查询，因此会产生执行成本，同时也提供了优化机会。

同时请注意，我们所有算法都使用立方体查询，而非数据库查询，作为基本工作单位。因此，这些算法对原始查询进行重写时，会利用层次多维空间中过滤器与分组器的组合，并且不会探索数据库管理系统内部的备选执行计划。

总体而言，整个设计还带来了另一条值得吸取的重要经验：在立方体查询层面，在数据库管理系统之外开展工作能带来显著的性能提升。

当然，将这些概念集成到数据库管理系统内部也值得探索，尽管这超出了本文的讨论范围。如果数据库管理系统内部或外部都不引入任何优化，那么执行 ANALYZE 最终就等同于执行 Min-MQO 算法，也就是依次批量执行五个数据库查询。除了在数据库管理系统外部进行优化，我们还可以考虑通过添加该算子来扩展数据库管理系统；在这种情况下，也可以相应扩展数据库管理系统的优化器。这既涉及此处列出的多查询优化方案，也涉及其他机会——例如，值得探索在这一情境下如何确定连接顺序。请注意，与传统的多查询优化算法不同，正是由于数据空间的特性，我们并不试图寻找重写方案，而是利用层次结构自动（因而高效地）执行重写。

6 实验

在本节中，我们通过实验评估所提出算法在不同数据和查询工作负载配置下的执行时间。在我们的全部实验中，我们使用 Delian Cubes 系统 [Del25], ，它是一个立方体查询应答引擎，我们已将算法集成其中。所有算法均以 Java 实现，作为 Delian Cubes 系统的一部分。我们使用 MySQL 8.0.34 作为底层数据库。就我们的评估而言，这些方法运行在一台 Windows 11、2.50 GHz、14 核处理器的系统上，该系统配备

32GB 主存和 1TB SSD。全部材料见：https://github.com/DAINTINESS-Group/DelianCubeEngine。

6.1 实验设置

实验目标和评估指标。 我们的评估主要旨在评估所提出的算法执行 ANALYZE 运算符的效率。因此，对于所有对手算法，我们测量完整计算 ANALYZE 查询答案所需的总执行时间，并分析各算法中不同促进器所耗费的时间。总查询执行时间 是各算法不同部分耗时之和，具体包括：(i) 解析 ANALYZE 查询的字符串输入表达式所需的解析时间，(ii) 构造辅助查询所需的构造时间，(iii) 执行辅助查询所需的促进器执行时间，(iv) 对辅助查询的结果进行后处理时间，并正确填充 ANALYZE 查询结果所需的时间。

对比算法。 在实验中，我们比较了三种不同的算法，评估它们实现 $ANALYZE$ 算子的总执行时间及其细分情况。为方便读者，我们在表 3 和下文的简要总结中概述这三种方法。所有算法的输入都是一个 ANALYZE 查询，其中包含两个分组器、一个可分配聚合函数和至少两个选择条件。回顾一下，该算子的语义要求执行 5 个内部辅助 立方体查询，即原始查询 q^{org}、两个同级查询 q^{s^A}，q^{s^B}，以及两个向下钻取立方体查询 q^{dd^A}，q^{dd^B}。

- $Min\text{-}MQO$ 是执行算子底层立方体查询合并程度最低的算法（因此称为“最小多查询优化”）。如前所述，该算法按照算子的语义构造五个辅助查询。实际上，这就是不进行任何优化、直接执行算子的简单方式，因此也无需对执行查询所得的结果进行后处理。

- $Max\text{-}MQO$ 是将辅助查询合并为单个辅助查询、合并程度最高的算法（因此称为“最大多查询优化”），并额外使用四个分组器来覆盖兄弟查询和下钻结果，同时替换选择条件（将输入原子替换为其父级和父值）。对结果进行后处理，并根据算子的语义将元组分配到正确的占位符。

- $Mid\text{-}MQO$ 是一种算法，在将辅助查询完全合并为一个查询和完全不合并这两个极端之间取得平衡。

具体而言，该算法构建三个辅助查询：(i) 两个同级查询，分别对应一个分组维度；(ii) 一个合并的原始查询与下钻查询多查询，它由原始查询和下钻查询组合而成。该算法利用了这些查询具有完全相同的选择条件，因而会搜索事实表中相同的元组数据空间这一特点。与前一种情况一样，生成的元组经过后处理后会分发到正确的占位符。

Name	# 可执行查询	后处理
Min-MQO	5	No
Mid-MQO	3	Yes
Max-MQO	1	Yes

表 3：方法详情

数据集。 我们使用表 4 中列出数据集进行评估。具体而言，这些数据集包括：(i)*Northwind* [Mic23], ，这是一个合成数据集，用于表示一家食品进出口公司的销售额和运营；(ii) *Foodmart*，这是*Foodmart* [Hyd25], 的立方体化版本，其中包含有关零售超市销售活动的合成数据；(iii)*pkdd99+*，这是*pkdd99* [ZYO99], 的扩展版本，是一个包含贷款和交易数据的金融数据集。我们为此创建了一个包含 1 亿个条目的事实表；以及 (iv) *TPC-DS* [TPC24], ，这是一项旨在评估分析平台性能的行业标准基准。我们使用缩放因子为 1、3.5 和 35 的三个版本，对算法的可扩展性进行了测试。由于 Delian Cubes 面向数据立方体运行，因此我们不直接使用数据集的原始模式，而是将其转换为经过清洗的立方体化星型模式版本，以支持层次结构和立方体查询。

Name	缩放因子	# 维度表	# 事实
Northwind	1	4	2K
Foodmart	1	5	288K
pkdd99+	166000	3	100M
TPC-DS	1,3.5,35	6	2M,10M,100M

表 4：数据集基本特征

查询工作负载。 查询工作负载涵盖了多种 ANALYZE 查询场景。ANALYZE 查询的特征包括：(i) 选择率、(ii) 过滤器/分组器取值的层级以及 (iii) 过滤器数量。选择率是指事实表中参与最终查询结果计算的元组所占的百分比。过滤器/分组器取值的层级

查询 ID	过滤级别	分组级别	筛选器数量	选择率			
				q^{org}	q^{sib_1}	q^{sib_2}	q^{A}
1	中高	中高	2	1%	8%	3%	20%
2	中高	中高	2	1%	4%	4%	9%
3	低高	低高	2	1.5%	8%	6%	20%
4	中高	中高	2	2%	4%	8%	20%
5	中高	中高	2	3%	8%	9%	20%
6	中高	中高	2	3.5%	8%	3%	20%
7	高-高	高-高	2	6%	30%	20%	100%
8	高-高	高-高	2	6.5%	30%	20%	100%
9	高-高	高-高	2	8%	37%	20%	100%
10	高-高	高-高	2	8%	37%	20%	100%

表5：TPC-DS 时间工作负载特征

查询 ID	过滤级别	分组级别	筛选器数量	选择率			
				q^{org}	q^{sib_1}	q^{sib_2}	q^{A}
1	低-低	低-低	2	0.001%	0.001%	0.35%	1%
2	中低	中低	2	0.002%	0.01%	0.35%	2%
3	高低	高低	2	0.01%	0.03%	1%	10%
4	低高	低高	2	0.2%	0.5%	2%	5%
5	低高	低高	2	0.35%	0.5%	2%	9%
6	中高	中高	2	0.5%	1%	3.5%	20%
7	中高	中高	2	0.9%	2%	6%	20%
8	中高	中高	2	1%	10%	20%	20%
9	高-高	高-高	2	2%	2%	9%	100%
10	高-高	高-高	2	2%	9%	10%	100%

表6：TPC-DS Item 工作负载特征

查询 ID	过滤级别	分组级别	筛选器数量	选择率			
				q^{org}	q^{sib_1}	q^{sib_2}	q^{A}
1	低-低	低-低	2	0.01%	0.01%	0.2%	0.2%
2	中低	中低	2	0.01%	0.25%	0.15%	3%
3	中低	中低	2	0.2%	0.2%	2%	2%
4	高低	高低	2	0.6%	3%	4%	20%
5	高-高	高-高	2	1.4%	17%	11%	100%
6	高-高	高-高	2	1.7%	17%	9%	100%
7	高低	高低	2	2%	2%	13%	13%
8	中高	中高	2	3%	17%	18%	100%
9	高-高	高-高	2	3%	17%	18%	100%
10	高-高	高-高	2	3%	17%	18%	100%

表7：pkdd99+ 工作负载特征

查询 ID	过滤级别	分组级别	筛选器数量	选择率			
				q^{org}	q^{sib_1}	q^{sib_2}	q^{A}
1	低-低	低-低	2	0.001%	1%	0.01%	3%
2	低-低	中-中	2	0.001%	1%	0.01%	3%
3	低-低	高-高	2	0.001%	1%	0.01%	3%
4	中-中	低-低	2	3%	9%	9%	30%
5	中-中	中-中	2	3%	9%	9%	30%
6	中-中	高-高	2	3%	9%	9%	30%
7	高-中	低-低	2	9%	30%	19%	67%
8	高-高	低-低	2	30%	30%	67%	100%
9	高-高	中-中	2	30%	30%	67%	100%
10	高-高	高-高	2	30%	30%	67%	100%

表 8：Foodmart 工作负载特征

查询 ID	过滤级别	分组级别	筛选器数量	选择率			
				q^{org}	q^{sib_1}	q^{sib_2}	q^{A}
1	低-低	低-低	2	0.5%	6%	1%	13%
2	低-低	中-中	2	0.5%	6%	1%	13%
3	低-低	高-高	2	0.5%	6%	1%	13%
4	中-中	低-低	2	1%	13%	1%	17%
5	中-中	中-中	2	1%	13%	1%	17%
6	高-中	低-低	2	6%	28%	13%	74%
7	高-中	高-高	2	6%	28%	13%	74%
8	高-高	低-低	2	13%	74%	17%	100%
9	高-高	中-中	2	13%	74%	17%	100%
10	高-高	高-高	2	13%	74%	17%	100%

表 9：Northwind 工作负载特征

values 是对其维度层次结构中筛选器/分组器级别高度的相对描述。表 5、6、7、8、9 总结了各数据集的工作负载特征。为评估算子在事实表上面对不同选择率时的表现，对于每个数据集，我们使用由 10 个查询组成的工作负载，其中 (a) 改变筛选器/分组器的各个层级，并且 (b) 使用一个大型和一个小型维度表。对于 TPC-DS 数据集，为了同时评估维度表大小的影响，我们使用两个查询工作负载，它们都涉及大型 日期 维度：(i) 使用大型 时间维度表的工作负载，以及 (b) 使用小型 商品 维度表的工作负载。在所有工作负载中，随着查询 ID 的增加， q^{org} 在所有数据集中也会增加。这两个工作负载的差异仅在于维度表大小不同所导致的选择率差异。时间 工作负载涉及两个相对较大的维度表，而 商品 工作负载则涉及一个大型和一个小型维度表。两个工作负载都涉及大型 日期 维度。对于其余数据集，我们使用了单个由 10 个查询组成的工作负载，其中包含不同层级的筛选器/分组器

这些查询涉及一个大型和一个小型维度表。为确保每个 ANALYZE 查询的结果大小相对较小，以便于理解和管理，我们根据第 3 节中讨论的约束，将每个 ANALYZE 查询的分组器层级设为尽可能高的级别。

6.2 查询特征对算法性能的影响

在本小节中，我们讨论各项查询特征对对抗算法整体性能的影响。在所有展示的图中，每个柱形表示总执行时间：Min-MQO 显示为深蓝色柱形，Mid-MQO 显示为蓝色柱形，Max-MQO 显示为浅蓝色柱形，超时限值则以红色虚线表示。在接下来的实验中，我们将评估各项特征对不同算法性能的影响。为此，我们定义一个参照查询，并在保持其他特征不变的情况下调整所评估特征的取值。我们使用了包含 1 亿个事实的 *TPC-DS* 数据集。查询特征的参照值为：20% 的选择率、中等分组器层级，以及每个查询 2 个过滤器。每项待测试查询特征的取值将在相应小节中给出。

6.2.1 选择率的影响

在本实验中，我们评估了 ANALYZE 查询的选择率如何影响三种对手算法的执行时间。图 2 展示了 ANALYZE 查询的选择率对所有方法的查询执行时间的影响。我们在实验中采用的选择率分别为事实表元组数量的 10%/50%/90%。对于所有对手算法，随着处理的元组数量增加，查询执行时间也随之增长。

请注意，在所有这些设置下，Mid-MQO 的表现始终优于对手方法。在选择率较低时，Min-MQO 和 Max-MQO 的表现接近，因为待处理的元组数量较少。Min-MQO 和 Mid-MQO 随选择率变化的扩展趋势相似；另一方面，随着选择率提高，Max-MQO 的性能迅速下降（请记住，Max-MQO 在内部需要处理最大的中间结果集）。

6.2.2 原子筛选条件数量的影响

在本实验中，我们评估了 ANALYZE 查询的选择条件中的原子筛选条件数量，如何影响三种对比算法的执行时间。图3展示了每个查询的过滤器数量对所有方法的查询执行时间的影响。评估中使用的原子筛选条件数量为 2/3/4/5。需要注意的是，连接操作的存在与过滤器数量直接相关：过滤器越多，需要连接的维度表也越多。自然，过滤器数量也会直接影响选择率，因为选择条件中的原子数量正在增加

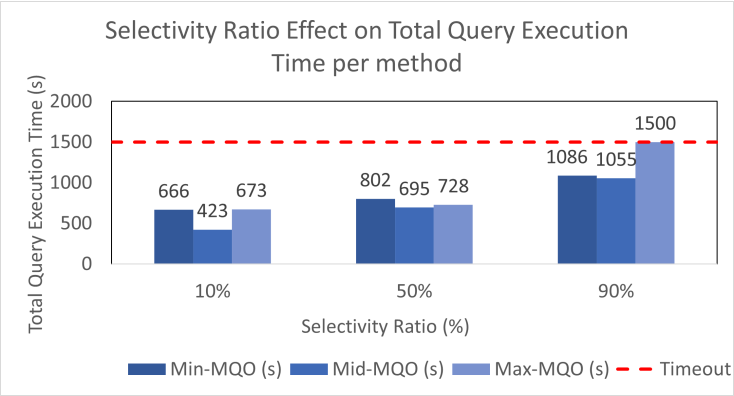


图 2：选择率对查询总执行时间的影响。*Max-MQO* 在选择率为 90% 的情况下未能完成执行。

。这是由查询本身的特性决定的，这意味着尽管增加了更多连接，但辅助查询处理过程中涉及的元组数量抵消了连接处理的影响。观察图3，可以清楚地看到几点：

- 如前所述，涉及的元组数量减少会直接缩短所有算法的执行时间。

- 随着选择率提高、数据量减少，Max-MQO 的性能提升速度远快于其他算法，这得益于其对数据库的单次访问。最终，在涉及的详细事实元组数量非常少时，Max-MQO 表现最佳。

- 与 Max-MQO 不同，筛选条件数量的变化对 Min-MQO 和 Mid-MQO 的影响并不相同。这两种方法会多次访问数据库，并针对多个查询执行多次连接操作，以处理额外的筛选条件。由于元组数量仍处于较高水平，与只访问数据库一次的 Max-MQO 相比，额外连接的需求削弱了这些元组所带来的收益。除一种情况外，Min-MQO 的性能始终最差。

6.2.3 其他考量

这里简要总结几个结果。

- 分组器聚合层级的影响。我们调整了各自层次结构中分组器层级的“高度”。为此，我们将层级的“高度”分为低 （层次结构中的最低层级）、高 （最顶层的全部层级及其直接子级）和中 （其他情况）。实验表明，分组器的聚合层级对总执行时间没有影响（图4）。

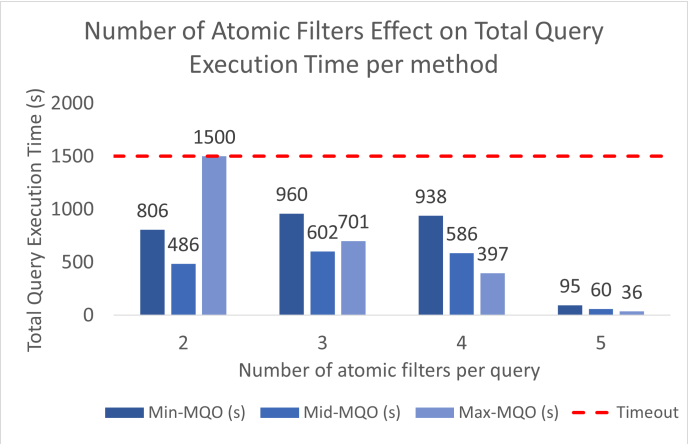


图3：原子筛选条件数量的影响。*Max-MQO* 在包含 2 个原子筛选条件时未能完成执行。

- 时间分解。我们分析了替代算法的内部行为，以找出可能的延迟点。为此，我们将查询执行过程划分为四个阶段，具体为：(a) 解析，(b) 构造辅助协调器查询，(c) 执行所有辅助查询所需的时间，以及(d) 对中间结果进行后处理，以构造最终查询答案（Min-MQO 无需此步骤）。为了涵盖原始查询对事实表覆盖范围的整个区间，我们将原始查询的选择率设为事实表元组的 10%、50% 和 90%。将总执行时间分解为上述四个阶段后发现，在所有情况下，协调器执行时间约占查询总执行时间的 99%，因此是所有优化工作的关键所在（图5）。

6.3 扩展对性能的影响

为了评估数据规模对算法性能的影响，我们需要让系统处理一组具有不同特征的查询工作负载。我们使用表5和表6中列出的 TPC-DS 项目 和时间 工作负载，并加入一组具有各种特征组合的 ANALYZE 查询。

图6a、6b、6c展示了在 *Item* 工作负载下，数据规模扩展对所有方法的查询执行时间的影响。Min-MQO以深蓝色表示，Mid-MQO以蓝色表示，Max-MQO以浅蓝色表示。评估中使用的同一数据集的事实表条目数分别为 200 万、1000 万和 1 亿。显然，随着数据规模扩大，所有算法的查询执行时间都会增加。请注意，随着事实表条目数增加，Max-MQO 的性能显著下降，且它与其他对手算法之间的性能差距不断扩大。Min-MQO 的性能与

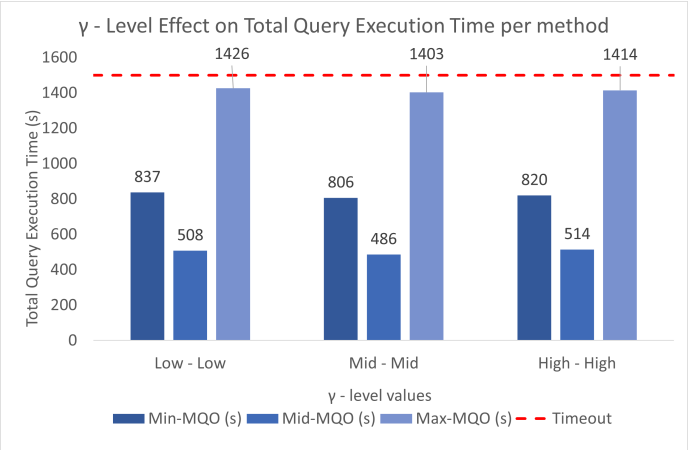


图4：分组器对总执行时间的影响

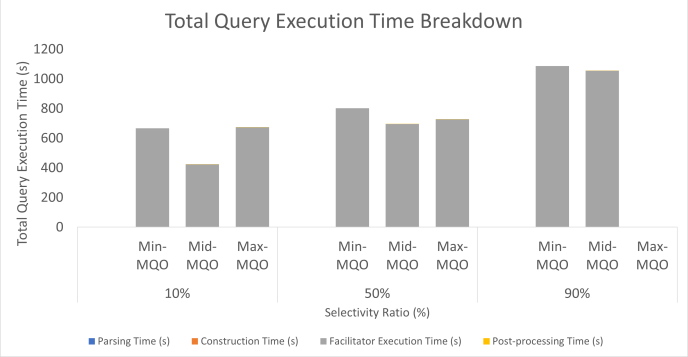
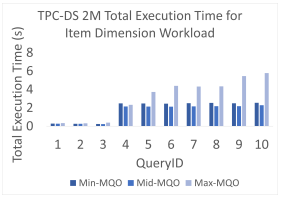


图5：总执行时间细分

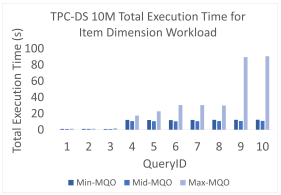
Mid-MQO 接近，但 Mid-MQO 始终更快。图 6g、6h、6i展示了在 时间工作负载下，数据规模扩展对所有方法查询执行时间的影响。显然，随着数据规模扩大，所有算法的查询执行时间都会增加。在这种情况下，维度表较大，同级促进器查询不会主导 ANALYZE 查询的执行时间。我们观察到，与其对手算法相比，Max-MQO 的性能随数据规模扩展得更好，并且在几乎每个查询中都是最快的算法。Min-MQO 的性能与 Mid-MQO 接近，但 Mid-MQO 始终更快。

6.4 性能评估

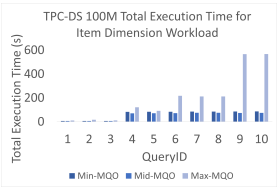
为了评估对抗方法的性能，我们在不同数据集和数据规模上评估了每种算法的性能。图 6 展示了



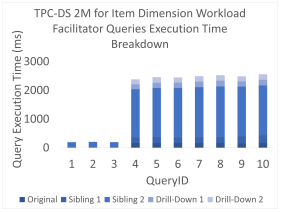
(a) TPC-DS 2M *Item* 工作负载



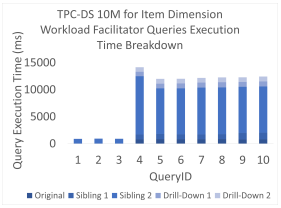
(b) TPC-DS 10M *Item* 工作负载



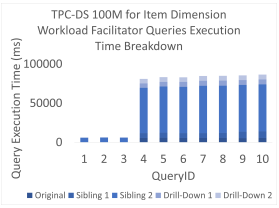
(c) TPC-DS 100M *Item* 工作负载



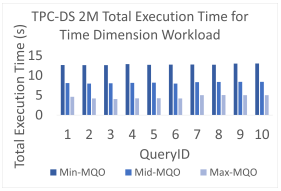
(d) TPC-DS 2M *Item* 工作负载 协调器查询细分



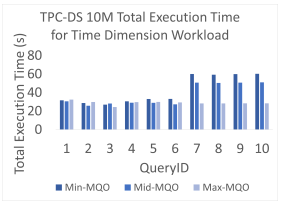
（e）TPC-DS 10M *Item* 工作负载 协调器查询细分



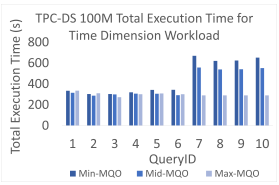
(f) TPC-DS 1亿 *Item* 工作负载 协调器查询细分



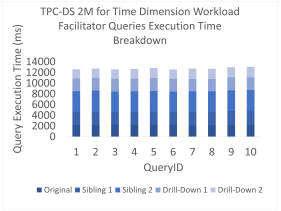
(g) TPC-DS 200万 *Time* 工作负载



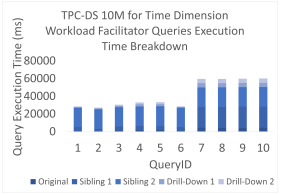
(h) TPC-DS 10M *Time* 工作负载



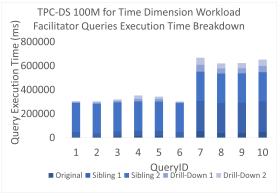
(i) TPC-DS 100M *Time* 工作负载



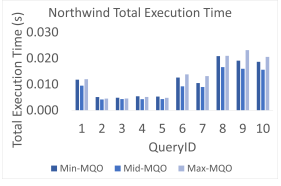
(j) TPC-DS 200万 *Time* 工作负载 协调器查询细分



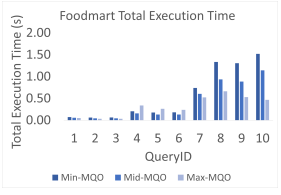
(k) TPC-DS 10M *Time* 工作负载 协调器查询细分



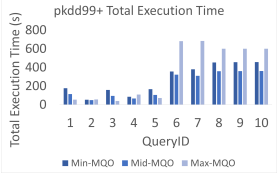
(l) TPC-DS 100M *Time* 工作负载 协调器查询细分



(m) Northwind 工作负载



(n) Foodmart Time 工作负载



(o) pkdd99+ 时间 工作负-load

图 6：性能评估 图 32

针对每个数据集，评估各算法的总执行时间。对于 TPC-DS $Item$ 和 $Time$ 工作负载，图 6 还展示了该工作负载中每个 ANALYZE 查询所涉及的辅助查询细分情况。

6.4.1 TPC-DS$Item$ 工作负载

图 6a、6b 和 6c 展示了在 TPC-DS 数据集的不同规模下，$Item$ 查询工作负载在所有方法中的执行时间。Min-MQO 以深蓝色线条表示，Mid-MQO 以蓝色线条表示，Max-MQO 以浅蓝色线条表示。该工作负载涉及的维度表为$Date$ 维度（≈80000个元组）和$Item$ 维度（≈ 20000 个元组）。我们观察到以下情况：

- 无论数据集规模如何，Mid-MQO 的表现都是最佳的。
- Max-MQO 的表现始终最差，尤其是在选择率不断增大时。
- Min-MQO 和 Mid-MQO 的执行时间没有显著差异。

图6d、6e、6f给出了该工作负载的ANALYZE查询中每个促进查询的详细执行时间。我们观察到，与Min-MQO相比，Mid-MQO未执行的下钻查询对总执行时间的影响并不显著。因此，Min-MQO与Mid-MQO之间的性能差异相对较小。与其他对手算法相比，Max-MQO处理的元组数量很大，在此背景下其表现持续较差。

6.4.2 TPC-DS时间 工作负载

图6g、6h、6i展示了时间 查询工作负载在每种TPC-DS数据集规模变化下、所有方法中的执行时间。该工作负载涉及的维度表为日期 维度（≈80000 元组）和时间维度（≈ 70000元组）。我们观察到以下几点。

- 在TPC-DS 2M中，Max-MQO的表现优于其他算法，而Min-MQO在所有情况下都是表现最差的算法。
- 在 TPC-DS 10M 和 100M 数据集中，当选择率较小时，随着选择率增加，Max-MQO 的性能几乎保持稳定；而 Min-MQO 和 Mid-MQO 的性能则会下降。

在此，由于两个维度表的规模都很大，Max-MQO 的性能保持稳定，因为它始终只对事实表的一个较大子集探索一次，并在选择率上升时避免了其他算法的兄弟查询所带来的沉重额外成本（详见 图 6j、6k、6l 中的分解，尤其是查询 ID 较大的高选择率查询）。辅助查询中执行时间的分解如 图 6j、6k、6l 所示，表明随着

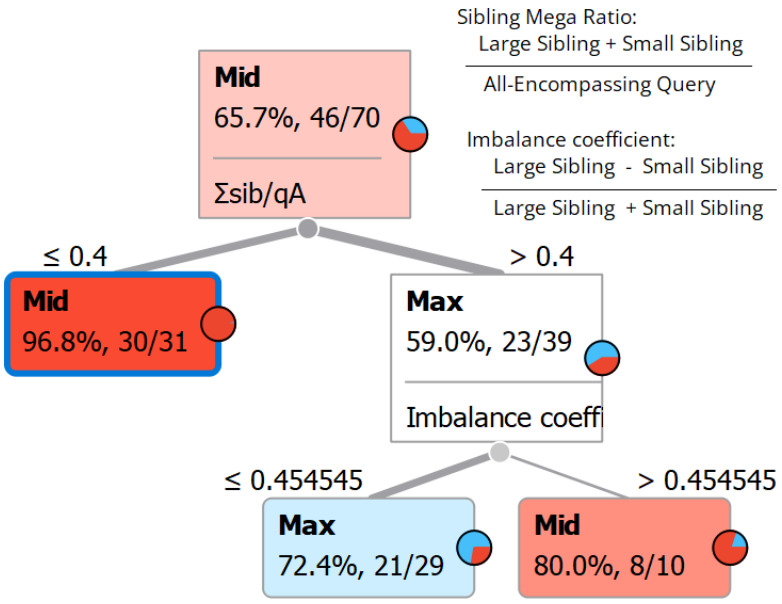


图 7：最优算法标准

选择率增加，下钻查询的执行成本解释了 Min-MQO 与 Mid-MQO 之间总执行时间差异的增大，从而也加大了 Min-MQO 与 Mid-MQO 之间的性能差异。

6.4.3 其他数据集

最后，在 3 个其他数据集（图 6m、6n、6o）上比较这些算法时，我们观察到：

- 在 Northwind 工作负载（表 9）中，涉及两个较小的维度表（≈90 元组和 ≈9 元组），Mid-MQO 始终胜过其他对手算法，而 Max-MQO 的表现不佳。

- 在 Foodmart 工作负载（表 8）中，涉及不平衡的维度表（≈10000 元组和 ≈1000 元组），Max-MQO 是最快的算法，而 Min-MQO 则最慢。

- 在 pkdd99+ 工作负载（表7）中，涉及一个较大的维度和一个较小的维度（≈30000 元组和 ≈5000），在高选择率下，Mid-MQO 的表现优于其他对手算法；而在低选择率下，Max-MQO 表现更好。

6.5 辅助查询对性能的影响

图6d、6e、6f 展示了 TPC-DS *Item* 工作负载中全部五个辅助查询的执行时间。我们观察到，无论数据集大小如何，Min-MQO 的表现都与 Mid-MQO 的表现非常接近（图6a、6b、6c）。TPC-DS Item 工作负载中下钻查询的执行时间占辅助查询总执行时间的比例很小。然而，兄弟查询的执行时间占总执行时间的比例非常大。由于这两种算法都会执行兄弟查询，而 Mid-MQO 省去了下钻查询的执行，因此显然 Min-MQO 的表现会与 Mid-MQO 接近。

图 6j、6k、6l 展示了 TPC-DS 时间 工作负载下全部五个辅助查询的执行时间。在 200 万条事实的情况下，Min-MQO 与 Mid-MQO 之间的性能差距显著。我们观察到，下钻查询的执行时间几乎占总执行时间的 1/3。在这种情况下，可以认为，不执行下钻查询的 Mid-MQO 相较于 Min-MQO 具有性能优势。在其他所有数据规模（1000 万和 1 亿）下，下钻查询的执行时间都不如 200 万条时那么长，Min-MQO 与 Mid-MQO 之间的性能差距也在缩小。

综合以上观察，我们可以认为，下钻查询的执行时间会影响 Min-MQO 与 Mid-MQO 之间的性能差距。如果下钻查询的执行时间占总执行时间的比例较大，那么 Mid-MQO 的性能会显著优于 Min-MQO。然而，如果下钻查询的执行时间占比不高，那么 Min-MQO 的性能虽不及 Mid-MQO，但与 Mid-MQO 相差不大。

这种情况何时会发生？这种区别的关键不在于向下钻取查询，而在于兄弟查询的覆盖范围。向下钻取查询所探索的详细事实元组空间，与原始查询的完全相同；不同的只是最终的聚合层级。另一方面，根据涉及的兄弟查询，兄弟查询可能只覆盖一小部分详细事实，也可能覆盖很大一部分。因此，它们与向下钻取查询的比例也会发生变化。

6.6 算法选择

Mid-MQO 作为查询处理策略是一种稳妥的选择，因为它既稳定，又在大多数情况下胜出。不过，我们能否超越这一初步结果？我们分析了大型 TPC-DS 和 pkdd 的全部 70 个工作负载（表 5、6、7）$99+$ 数据集（7 个工作负载，每个包含 10 个查询），以便通过易于计算的成本度量准确判断每种查询处理策略何时胜出。最终（图 7），当 *(a)* 兄弟查询触及的事实元组数量相对于全涵盖查询而言相当多（超过 *40%*），且 *(b)* 兄弟查询之间的不均衡程度相当低（低于 *45%*）时，*Max-MQO* 胜出。其他所有情况下，*Mid-MQO* 均胜出。理由很简单：

1. 当 q^A 相对于辅助查询过大时，它会探索过多的数据空间，因此 Max-MQO 会落败。2. 当兄弟查询都相当大且彼此不均衡时，它们的重叠并不显著，因此 Max-MQO 虽然只需访问一次，却无法节省时间。

Max-MQO 能够节省时间的唯一情况，是规模较大的兄弟查询存在重叠（通过不平衡系数估算）。当然，确切的阈值属于工程默认值，还需随着时间推移进行更广泛的评估；不过，其基本原理与 q^A 的理论设置一致。

7 结论

本文介绍了 ANALYZE，这是一种新颖的意图立方体查询算子，通过在一次调用中结合原始查询、兄弟查询和下钻查询，并赋予其形式语义，为数据提供 360° 视图。我们已经证明，该算子的内部查询可以安全地进行合并，并具有理论上的正确性保证，从而能够在算子层面实施有原则的多查询优化，而无需修改底层查询优化器。我们还在数据分析系统中引入并实现了三种执行策略，并进行了广泛评估。结果表明，部分合并（Mid-MQO）能够在各种工作负载下提供稳健且可扩展的性能；完全合并（Max-MQO）仅在兄弟查询规模较大且存在显著重叠时才有益；而未优化策略（Min-MQO）始终表现欠佳。

作为未来工作的组成部分，可以沿着若干方向继续探索。目前，普通数据库管理系统只会执行 Min-MQO，以处理该算子的辅助查询集合。由此得到的一个启示是，外部数据分析引擎可以 在数据库管理系统之外优化查询执行。这一经验也适用于未来的意图算子（如 PREDICT、EXPLAIN 等）。利用该算子的形式化代数特性，也引出了将其与基于代价的数据库优化器或 Apache Spark 等平台集成的问题。若干可能的扩展涉及当前版本 ANALYZE 的内部设计选择。放宽部分约束（例如，增加分组器，或采用基于用户定义或上下文相关基准的同级项 [SPV07, MB25]）也有待研究。例如，可以将我们的结果推广到两个以上的分组器；我们推测，尽管这会带来形式化方面的额外开销，但参与 ANALYZE 查询构造的分组器数量并无上限。再如，也可以规范甚至自动化用于与原始查询进行对比的基准（例如，将某一季度的销售额与上一年的同一季度进行比较），这也会对相应的查询处理算法产生影响。

致谢。 本研究项目是在希腊研究与创新基金会“第三轮希腊研究与创新基金会研究项目征集——支持教师与研究人员”（希腊研究与创新基金会项目编号：23640）的框架下开展的。

D. Gkitsakis 协助完成了代码的早期试用版本。

8 工件

源代码、数据和/或其他工件已发布于 https: // github.com/DAINTINESS-Group/DelianCubeEngine。请访问 https://tinyurl.com/bddf4tzv 中的程序包 analyzemqoexperiments，其中包含一个 Markdown 文件，说明如何重复这些实验。

参考文献

[AKS+21] Firas Abuzaid、Peter Kraft、Sahaana Suri、Edward Gan、Eric Xu、Atul Shenoy、Asvin Ananthanarayan、John Sheu、Erik Mei-jer、Xi Wu、Jeffrey F. Naughton、Peter Bailis 和 Matei Zaharia。DIFF：用于大规模数据解释的关系型接口。*VLDBJ.*，30(1):45–70，2021年。[Ame24] Sihem Amer-Yahia。用于数据探索的智能体。*Proc. VLDB Endow.*，17(12):4521–4530，2024年。[AMP23] Sihem Amer-Yahia、Patrick Marcel 和 Ver´onika Peralta。面向大众的数据叙述：挑战与机遇。载于第*26*届数据库技术拓展国际会议论文集，*EDBT 2023*，希腊约阿尼纳，*2023*年*3*月*28*日至*31*日，第855–858页。OpenProceedings.org，2023年。[BRH+22] Tijl De Bie、Luc De Raedt、Jos´e Hern´andez-Orallo、Holger H. Hoos、Padhraic Smyth 和 Christopher K. I. Williams。实现数据科学自动化。*Commun. ACM*，65(3):76–87，2022年。[Del25] Delian Cubes。https://github.com/DAINTINESS-Group/ DelianCubeEngine，2025年。约阿尼纳大学 DAINTINESS GROUP。[DHX+19] Rui Ding、Shi Han、Yong Xu、Haidong Zhang 和 Dongmei Zhang。QuickInsights：从多维数据中快速、自动地发现洞察。载于*SIGMOD* 论文集，第317–332页，2019年。[EMS19] Ori Bar El、Tova Milo 和 Amit Somech。ATENA：一种基于深度强化学习的自主数据探索系统。载于第*28*届 *ACM* 国际信息与知识管理会议论文集，*CIKM 2019*，中国北京，*2019*年*11*月*3*日至*7*日，第2873–2876页，2019年。[FGM+21] Matteo Francia、Matteo Golfarelli、Patrick Marcel、Stefano Rizzi 和 Panos Vassiliadis。评估用于数据立方体交互分析的查询。载于第*24*届数据库技术拓展国际会议论文集，*EDBT 2021*，塞浦路斯尼科西亚，*2021*年*3*月*23*日至*26*日，第121–132页，2021年。

[FGM+23] Matteo Francia、Matteo Golfarelli、Patrick Marcel、Stefano Rizzi 和 Panos Vassiliadis。为多维数据交互分析推荐评估查询。*IEEE Trans. Knowl. Data Eng.*，35(6):6421–6434，2023年。[FMPR22] Matteo Francia、Patrick Marcel、Ver´onika Peralta 和 Stefano Rizzi。利用模型增强数据立方体，以描述多维数据。*Inf. Syst. Frontiers*，24(1):31–48，2022年。[FRM24] Matteo Francia、Stefano Rizzi 和 Patrick Marcel。通过意图分析解释数据立方体度量。*Inf. Syst.*，121:102338，2024年。[GV13] Dimitrios Gkesoulis 和 Panos Vassiliadis。Cinecubes：轻松打造明星般的数据立方体。载于第*16*届数据仓库与联机分析处理国际研讨会论文集，*DOLAP 2013*，美国加利福尼亚州旧金山，*2013*年*10*月*28*日，第3–10页，2013年。[GVM15] Dimitrios Gkesoulis、Panos Vassiliadis 和 Petros Manousis。Cinecubes：帮助数据工作者从在线分析处理查询中获得洞察。*Inf. Syst.*，53:60–86，2015年。[HRK+09] Mingsheng Hong、Mirek Riedewald、Christoph Koch、Johannes Gehrke 和 Alan J. Demers。基于规则的多查询优化。载于*EDBT 2009*：第*12*届数据库技术拓展国际会议论文集，俄罗斯圣彼得堡，*2009*年*3*月*24*日至*26*日，第120–131页，2009年。[Hyd25] Julian Hyde。Foodmart。https://github.com/julianhyde/foodmart-data-hsqldb，2025年。访问日期：2025-07-24。[IPC15] Stratos Idreos、Olga Papaemmanouil 和 Surajit Chaudhuri。数据探索技术综述。载于*2015*年 *ACM SIGMOD* 国际数据管理会议论文集，第277–281页，2015年。[KS15] Tarun Kathuria 和 S. Sudarshan。Greedy awakens：高效且可证明的多查询优化。*CoRR*，abs/1512.02568，2015年。[LKDL12] Wangchao Le、Anastasios Kementsietsidis、Songyun Duan 和 Feifei Li。面向 SPARQL 的可扩展多查询优化。载于*IEEE* 第*28*届数据工程国际会议（*ICDE 2012*），美国华盛顿哥伦比亚特区（弗吉尼亚州阿灵顿），*2012*年*4*月*1*日至*5*日，第666–677页，2012年。[LMS+25] Tavor Lipman、Tova Milo、Amit Somech、Tomer Wolfson 和 Oz Zafar。LINX：一种由语言驱动、面向目标的自动数据探索生成式系统。载于第*28*届数据库技术拓展国际会议论文集，*EDBT 2025*，

西班牙巴塞罗那，*2025*年*3*月*25*日至*28*日，第270–283页。
OpenProceedings.org，2025年。[LYZ+23] Haotian Li、Lu Ying、Haidong Zhang、Yingcai Wu、Huamin Qu 和 Yun Wang。Notable：计算笔记本中用于数据叙事的即时助手。载于 *CHI*，2023年。[MB25] Pablo Mateos 和 Alejandro Bellog´ın。上下文感知推荐系统近期进展的系统性文献综述。《人工智能评论》，58(1):20，2025年。[MDHZ21] Pingchuan Ma、Rui Ding、Shi Han 和 Dongmei Zhang。Metainsight：自动发现用于探索性数据分析的结构化知识。载于 *SIGMOD '21*：国际数据管理会议，中国线上会议，*2021*年*6*月*20*日至*25*日，第1262–1274页。ACM，2021年。[MDW+23]Pingchuan Ma、Rui Ding、Shuai Wang、Shi Han 和 Dongmei Zhang。Insightpilot：大语言模型驱动的自动化数据探索系统。载于 *EMNLP'2023*，2023年。[Mic23]Microsoft。Northwind。
https://github.com/microsoft/sql-server-samples/tree/master/samples/databases/northwind-pubs，2023年。访问日期：2025-09-14。[MS20] Tova Milo 和 Amit Somech。通过机器学习自动进行探索性数据分析：概述。载于*2020*年数据管理国际会议论文集，*SIGMOD 2020*会议，在线会议 [美国俄勒冈州波特兰]，*2020*年*6*月*14*日至*19*日，第2617–2622页，2020年。[PAB+21] Aur´elien Personnaz、Sihem Amer-Yahia、Laure Berti-´Equille、Max- imilian Fabricius 和 Srividya Subramanian。DORA THE EXPLORER：利用交互式深度强化学习探索超大型数据。载于*CIKM '21*：第*30*届 *ACM* 信息与知识管理国际会议，线上会议，澳大利亚昆士兰州，*2021*年*11*月*1*日至*5*日，第4769–4773页，2021年。[RS09] Prasan Roy 和 S. Sudarshan。多查询优化。载于《数据库系统百科全书》，第1849–1852页。施普林格美国，2009年。[SAM98] Sunita Sarawagi、Rakesh Agrawal 和 Nimrod Megiddo。数据驱动的 OLAP数据立方体探索。载于 数据库技术进展——*EDBT'98*，第*6*届数据库技术扩展国际会议，西班牙瓦伦西亚，*1998*年*3*月*23*日至*27*日，会议论文集，第168–182页，1998年。

[Sar99] Sunita Sarawagi。解释多维聚合之间的差异。载于*VLDB'99*，第*25*届超大型数据库国际会议论文集，*1999*年*9*月*7*日至*10*日，英国苏格兰爱丁堡，第42–53页，1999年。[Sar00] Sunita Sarawagi。用户自适应的多维数据探索。载于*VLDB 2000*，第*26*届超大型数据库国际会议论文集，*2000*年*9*月*10*日至*14*日，埃及开罗，第307–316页，2000年。[SCC+23] Mengdi Sun、Ligan Cai、Weiwei Cui、Yanqiu Wu、Yang Shi 和 Nan Cao。Erato：通过事实插值进行协作式数据故事编辑。*IEEE* 可视化、计算机图形学汇刊，29(1):983–993，2023年。[Sel88] Timos K. Sellis。多查询优化。*ACM* 数据库系统汇刊，13(1):23–52，1988年。[SG90] Timos K. Sellis 和 Subrata Ghosh。关于多查询优化问题。*IEEE* 知识与数据工程汇刊，2(2):262–266，1990年。[SPV07] Kostas Stefanidis、Evaggelia Pitoura 和 Panos Vassiliadis。上下文感知偏好数据库系统。《普适计算与通信国际期刊》，3(4):439–460，2007年。[SS01] Gayatri Sathe 和 Sunita Sarawagi。多维 OLAP数据中的智能汇总。载于*VLDB 2001*，第*27*届超大型数据库国际会议论文集，*2001*年*9*月*11*日至*14*日，意大利罗马，第531–540页，2001年。[SXS+21] Danqing Shi、Xinyue Xu、Fuling Sun、Yang Shi 和 Nan Cao。Calliope：从电子表格自动生成可视化数据故事。*IEEE* 可视化、计算机图形学汇刊，27(2):453–463，2021年。[THY+17] Bo Tang、Shi Han、Man Lung Yiu、Rui Ding 和 Dongmei Zhang。从多维数据中提取 top-k 洞察。载于*2017*年 *ACM* 数据管理国际会议论文集，*SIGMOD 2017* 会议，美国伊利诺伊州芝加哥，*2017*年*5*月*14*日至*19*日，第1509–1524页，2017年。[TPC24] TPC。TPC-DS。https://www.tpc.org/tpcds/，2024年。访问日期：2025-06-05。[Vas22] Panos Vassiliadis。带比较操作的立方体代数：包含、重叠、距离和可用性。*CoRR*，abs/2203.09390，2022年。[Vas23] Panos Vassiliadis。通过先前的立方体查询结果回答立方体查询。载于设计、优化、语言与分析处理国际研讨会第*25*届会议论文集

大数据（*DOLAP*），与第*26*届扩展数据库技术国际会议和第*26*届国际数据库理论会议（*EDBT/ICDT 2023*）同期举行，希腊约阿尼纳，*2023*年*3*月*28*日，第71–75页。CEUR-WS.org，2023年。[VM18] Panos Vassiliadis 和 Patrick Marcel。《通往亮点之路由善意铺就：展望联机分析处理建模的范式转变》。载于第*20*届大数据设计、优化、语言与分析处理国际研讨会论文集，与第*10*届*EDBT/ICDT*联合会议（*EDBT/ICDT 2018*）同期举行，奥地利维也纳，*2018*年*3*月*26*日至*29*日，2018年。[VMR19] Panos Vassiliadis、Patrick Marcel 和 Stefano Rizzi。《超越汇总与下钻：重塑联机分析处理的意图分析模型》。*Information Systems*，85：68–91，2019年。[WC13] Guoping Wang 和 Chee-Yong Chan。《MapReduce框架中的多查询优化》。《*VLDB*学报》，7(3)：145–156，2013年。[WSZ+20] Yun Wang、Zhida Sun、Haidong Zhang、Weiwei Cui、Ke Xu、Xiaojuan Ma 和 Dongmei Zhang。《Datashot：从表格数据自动生成事实表》。《*IEEE*计算机视觉与图形学汇刊》，26(1)：895–905，2020年。[XWJ24] Junjie Xing、Xinyu Wang 和 H. V. Jagadish。《多维数据的数据驱动洞察合成》。《*VLDB*学报》，17(5)：1007–1019，2024年。[ZYO99] Ning Zhong、Yiyu Yao 和 Setsuo Ohsuga。《面向特异性的多数据库挖掘》。载于《数据挖掘与知识发现原理，第三届欧洲会议，*PKDD '99*，捷克共和国布拉格，*1999*年*9*月*15*日至*18*日，论文集》，《计算机科学讲义》第1704卷，第136–146页。施普林格，1999年。