完整性保证：通过 U-表达式高效验证 SQL 等价性

Haoran Ding[1, 2, *], Zhaoguo Wang[1, †], Zuming Jiang[3], Haibo Chen[1], Jinyang Li[4], Ruzica Piskac[5]
[1]上海交通大学并行与分布式系统研究所 [2]麻省理工学院计算机科学与人工智能实验室 [3]香港大学 [4]纽约大学 [5]耶鲁大学

arXiv:2611.28516v1 [cs.DB] 21 Nov 2026

摘要

SQL 查询等价性的自动验证是数据库研究中的一个基础问题。近期研究开发了具有完备性保证的验证器，这些验证器能够生成反例以证明查询不等价，并适用于各种场景。然而，它们必须求解复杂公式，因此可能无法在合理的时间预算内生成反例。

本文介绍了一种名为 SQLution 的新型完备验证器，它不仅大幅缩短了验证时间，还支持更多 SQL 特性。其核心思路是使用 U-表达式（一种函数代数）表示 SQL 语义；与以往方法相比，U-表达式所需的公式简单得多，因此能够实现更高效的 SMT 求解。为了将这一思路转化为实用的验证器，SQLution 进一步解决了使用 U-表达式建模 SQL 时面临的两个技术挑战，即缺少元组排序和存在无界求和。具体而言，为支持元组排序，SQLution 提出了*排序语义编码*，将 U-表达式与一阶逻辑（FOL）结合起来。为处理无界求和，SQLution 引入*无界求和展开*，将其转换为有界求和，而有界求和可直接编码为一阶逻辑公式。与最先进的验证器相比，SQLution 支持从 Spider 得出的全部 271 个非等价查询对，并将验证时间最多缩短 98.6%。

1 引言

SQL 查询等价性的自动验证是数据库理论中的一个基础问题 [4]。如果两个查询在所有可能的数据库实例上都产生相同结果，则认为它们等价。开发此类等价性的验证器在广泛的应用场景中至关重要。例如，这类验证器可以帮助自动发现重写规则 [7, 22, 33, 42, 55]，将给定查询转换为另一个等价但速度更快的查询。验证器还有助于检测数据库错误 [30, 44]。某些错误会导致查询优化器将查询重写为不等价的查询 [30]。然而，SQL 等价性问题是不可判定的 [17, 20]，因此任何验证器都无法同时保证可靠性和完备性。

近年来，人们对构建完备验证器 [21, 30, 60] 的兴趣日益浓厚，这类验证器会自动搜索此类反例。这在多个重要场景中都很有用，例如数据库错误检测、基于示例的查询合成、查询消歧和 SQL 评分。相比之下，可靠验证器 [20, 21, 27, 54, 55, 61, 62] 通常无法提供反例，因为它们本质上并不完备，并且可能会错误地将等价查询判定为不等价。

最先进的完备验证器 [30, 60]将 SQL 表建模为列表，并提出基于列表的有界符号推理。具体而言，验证器为表大小设置一个界限 b，从而能够使用包含 b 符号元组的列表表示每个输入表。通过对这些符号列表上的 SQL 查询进行符号化求值，验证器可以系统地将查询编码为一阶逻辑（FOL）公式，从而能够使用 SMT 求解器 [8,9, 26] 生成符号元组的具体赋值，作为证明查询不等价的反例。

尽管前景可期，现有的完备验证方法效率低下：基于列表的编码会产生复杂的公式，导致 SMT 求解器运行时间较长。例如，对于 Spider 基准测试中 AST 节点总数为 29 的一对查询，即使表大小界限仅为 1，生成的公式也包含 2,916 个 AST 节点。当界限增至 3 时，公式中的 AST 节点数量会增加到 18,001 个。效率低下有两个根本原因。首先，符号执行会导致编码复杂。除了对输入表和最终查询结果进行建模之外，现有验证器还会为每个 SQL 运算符的中间结果生成一个符号列表，这显著增加了符号变量的数量。此外，对于查询结果中的每个元组，验证器都会生成一个独立公式来描述该元组如何由输入推导得出。随后，整体编码表示为所有此类公式的合取，从而产生包含大量逻辑和算术运算的表达式——

器。其次，使列表语义与 SQL 的包语义保持一致也会导致复杂的编码。在 SQL 的包语义下，表中的元组没有固定顺序，并且允许重复。因此，在包语义下比较查询结果时，现有验证器需要考虑列表中元组的所有可能排列，从而生成规模庞大且复杂的公式，给 SMT 求解器带来挑战。

为克服这些局限，我们的关键洞见是以轻量的方式对查询等价性进行编码，其基础是一种名为 U-表达式 [20] 的代数。基于这一洞见，我们开发了一种新的完备验证器 SQLution。现有的可靠验证器 SQLSolver [27] 表明，U-表达式能够对广泛的 SQL 特性进行建模，包括以往工作不支持的特性 [30, 60]。U-表达式将 SQL 查询表示为一个函数 $f(t)$，该函数返回查询结果中任意元组 t 的多重性。这种表示简洁地捕捉了查询语义，并能用大幅减少的变量和更简单的公式进行等价性推理。由于可以直接计算任意元组的多重性，SQLution 无需为每个输出元组分别生成公式。此外，查询等价性可以表示为

[*]本研究完成于 Haoran Ding 在上海交通大学攻读博士期间。他目前是麻省理工学院的博士后研究员。[†]通讯作者。

一种尤为简洁优雅的形式：当且仅当两个查询对应的 U-表达式对任意输入元组都产生相同的整数值时，这两个查询才等价。形式上，对于 U-表达式 $f_1(t)$ 和 $f_2(t)$，等价性检查可归结为证明公式 $\exists t.f_1(t) \neq f_2(t)$ 不可满足。与比较两个符号列表不同，比较两个整数值的编码要简单得多，也更容易高效求解。

尽管 SQLSolver [27] 已使用 U-表达式进行可靠验证，但其用于检查 U-表达式等价性的算法本质上并不完备，也无法为非等价查询生成反例。由于查询等价性验证是不可判定的，因此将 SQLSolver 扩展为完备验证器并非易事。因此，要实现基于 U-表达式的完备自动验证器，就必须解决 SQLSolver 无法处理的两个挑战：对有序包语义进行推理，以及在 U-表达式中编码无界求和。第一个挑战源于 U-表达式本身的语义：根据定义，U-表达式仅能捕获包语义，因此无法直接对具有有序包语义的查询建模。这些查询包含 ORDER BY、LIMIT 或 OFFSET [45], 等子句，而查询结果中元组的顺序十分重要。由于 U-表达式针对的是单个元组的多重性，而非元组之间的相对顺序，因此并未提供用于表达元组排序的内置机制。SQLSolver 通过句法匹配算法支持 ORDER BY，但该算法无法确保完备性。具体而言，它要求：1）两个查询包含相同数量的 ORDER BY 子句；2）一个查询中的每个 ORDER BY 子句，都能在另一个查询中找到一个匹配的 ORDER BY 子句，且两者的排序方向和属性相同；3）每对匹配的 ORDER BY 子句所对应的子查询彼此等价。由于语义等价的查询未必具有匹配的句法结构，因此该过程并不完备。未能找到这样的匹配，并不能证明查询不等价，也无法得出反例。由于 SQLution 必须对有序语义进行完备推理，因此无法沿用这一句法匹配算法。

为克服这一挑战，我们引入一种**排序语义编码** 技术来处理 ORDER BY、LIMIT 和 OFFSET。该技术通过编码查询语义而非进行语法比较来推理查询等价性，从而确保完备性。具体而言，它系统地重写 SQL 查询，以在保留查询语义的同时消除 LIMIT 和 OFFSET。一般情况下无法移除 ORDER BY，因为有序包语义无法归约为纯包语义。幸运的是，我们观察到，尽管 U-表达式只能表示查询结果中单个元组的多重性，但可以将其与一阶逻辑公式结合，显式编码元组之间的排序关系。因此，我们使用两个 U-表达式编码查询结果中存在元组 t_1 和 t_2。随后，我们基于一阶逻辑编码：在一个查询结果中 t_1 排在 t_2 之前，而在另一个查询结果中排序相反。如果上述约束可满足，则查询不等价，我们将使用 SMT 求解器生成一个反例。为简洁起见，在上下文明确时，将有序包语义简称为排序语义。

第二个挑战是，在确保完备性的同时，目前没有能够推理 U-表达式的判定过程。主要困难源于 U-表达式中的无界求和（Σ），因为这类求和难以直接编码为一阶逻辑公式。在无界求和 $\sum_t e(t)$ 中，变量 t 遍历的是所有可能的元组，而非静态有界的集合，这意味着求和域是无界的。要建模 PROJECTION 和聚合函数等 SQL 结构，离不开这类求和。然而，一阶逻辑公式无法直接表示无界表达式之和。SQLSolver 提出了一种基于近似的方法，用于编码含有无界求和的 U-表达式，但无法确保完备性。具体而言，编码所得的结果是一个过近似公式：若该公式不可满足，则将原始 U-表达式视为等价；但若该公式可满足，原始 U-表达式则可能等价，也可能不等价。因此，SQLSolver 的方法无法判定两个查询是否不等价，也无法生成反例。为确保完备性，SQLution 不能复用 SQLSolver 的编码算法。

为处理无界求和并提供完备性保证，我们提出了一种适用于 U-表达式的新验证算法，该算法扩展了经典的有界验证思想 [10, 32, 34–37, 46]。通过假设每个表都包含数量有界的不同元组，我们观察到，每个无界求和都等价于一个有界求和，而后者可以直接编码为一阶逻辑公式。直观地说，我们发现只有有限个元组能够使 $e(t)$ 产生非零值。因此，无需真正枚举无界求和域中的所有可能元组。基于这一观察，我们提出**无界求和展开**，为每个无界求和精确构造一个有界域，该域涵盖所有能够使 $e(t)$ 产生非零值的元组 t。随后，可以将无界求和展开为一个等价的有界求和。

我们在源自 Spider [58], 的 997 对 SQL 查询上评估了 SQLution。Spider 是一种广泛用于基于大语言模型的查询合成的基准测试 [40, 41, 48]。SQLution 成功处理了所有测试用例，其中包括 271 对非等价查询。相比之下，当前最先进的完备验证器 Polygon 仅支持 716 个用例，其中 162 个为非等价查询。对于两个验证器均判定为非等价的查询，SQLution 实现了显著的性能提升：在 92.9% 的情况下，验证时间减少了超过 50%；在 43.6% 的情况下，减少了超过 80%；最大时间缩减幅度达到 98.6%（从 802 ms 降至 11 ms）。

2 背景与动机

2.1 可靠验证器与完备验证器

SQL 查询等价性问题是不可判定的 [17, 20], ，这意味着验证器不可能同时保证**可靠性** 和**完备性**。可靠验证器（例如 SQLSolver）能够保证，它证明等价的每一对查询在所有数据库实例上确实等价。然而，由于可靠验证器固有的不完备性，即使某些查询确实等价，也可能无法通过验证。因此，可靠验证器通常无法在未能证明等价时给出反例。与可靠验证器不同，本文关注完备验证器，它们能够保证未通过验证的查询确实不等价。完备验证器会提供反例来证明查询不等价，即一组具体的数据库表

```
创建表 酒店(id 整数, 星级 整数, 价格 整数);Q1:(查询 * 来自 酒店
其中 星级 < 4)并集 ALL(查询 * 来自 酒店 其中 价格 < 500)
Q2: 查询 * 来自 酒店 其中 星级 < 4
```

图1：两个不等价 SQL 查询的示例。

在这些输入表上，两个查询会产生不同的结果。由于该问题不可判定，完备验证器和可靠验证器通常采用截然不同的算法。要将 SQLSolver 这类可靠验证器扩展为完备验证器，难以实现。为了生成反例来证明查询不等价，完备验证器通常采用有界验证方法。给定界限 b，完备验证器会搜索输入表，使每个表的大小都不超过该界限，并且两个查询产生不同的结果。如果存在这样的反例，验证器就能证明这些查询不等价。否则，当每个表的大小都不超过该界限时，验证器会判定这些查询等价。增大界限 b 可使验证器考虑更大的表。

2.2 SQL 等价性验证器的应用

SQL 等价性验证器在多种场景中都发现新的重写规则 [33, 55]。查询优化器通常会应用重写规则，将查询转换为另一个等价但速度更快的查询。已有研究 [33, 55] 通过先枚举可能的重写规则，再使用验证器识别重写前后能够确保查询等价的规则，从而自动发现新规则。完备验证器在多种场景中也很有用。首先，它们有助于检测和理解数据库优化器中的错误 [30, 44, 60]。尽管查询在构造上等价，优化器的实现仍可能存在错误 [1–3]。先前的研究 [30] 已在 Apache Calcite 和 MySQL 中发现此类错误。完备验证器生成的反例可以帮助开发者理解并定位错误。其次，完备验证器生成的反例可用于指导大语言模型重写 SQL 查询。例如，GenRewrite [43] 提出了一种反例引导方法，利用大语言模型将给定的 SQL 查询重写为等价但速度更快的查询。当重写后的查询与原查询不等价时，GenRewrite 会使用反例指导大语言模型修正重写结果。第三，完备验证器有助于评估大语言模型在文本到 SQL 任务中的能力。文本到 SQL 任务要求大语言模型将自然语言问题转换为 SQL 查询。常见的文本到 SQL 基准测试（例如 Spider）可以使用完备验证器检查生成的 SQL 查询是否与参考 SQL 查询等价。在这些场景中有效应用完备验证器不属于本文的讨论范围，仍是未来值得研究的课题。

2.3 基于列表的SQL等价性验证

我们通过图1中的示例说明基于列表的完备 SQL 等价性验证器如何运行。在此示例中，第一个查询使用 UNION ALL 将星级低于 4 的酒店与价格低于 500 的酒店连接起来，并对同时满足两个条件的元组进行重复计数。第二个查询只返回星级低于 4 的酒店。因此，这两个查询并不等价。我们从界限 1 开始。基于列表的方法按如下方式执行。验证器首先构造符号列表来表示输入表和查询输出。在界限为 1 时，表 hotel 的符号列表最多包含一个符号元组：$l_0 := [(del_0, id_0, star_0, price_0)]$。这里，布尔变量 del_0 表示元组删除（即，当且仅当 del_0 为真时，该元组不存在），而整数变量 id_0、$star_0$和 $price_0$ 对应于元组的属性。根据这一界限，Q_1 的结果最多包含两个元组，而 Q_2 的结果最多包含一个元组。因此，查询结果的符号列表定义为：Q_1对应的 $l_1 := [(del_1, id_1, star_1, price_1), (del_2, id_2, star_2, price_2)]$，以及 Q_2对应的 $l_2 := [(del_3, id_3, star_3, price_3)]$，其中 del_1、del_2和del_3 表示相应元组是否存在。

接下来，Q_1 的语义被编码为 F_1。

$$\begin{aligned} F_1 \coloneqq\ & (del_1 \leftrightarrow (del_0 \vee \neg(star_0 < 4))) \\ & \wedge (id_1 = id_0) \wedge (star_1 = star_0) \wedge (price_1 = price_0) \\ & \wedge (del_2 \leftrightarrow (del_0 \vee \neg(price_0 < 500))) \\ & \wedge (id_2 = id_0) \wedge (star_2 = star_0) \wedge (price_2 = price_0) \end{aligned}$$

所得公式是两个部分的合取。第一部分编码第一个子查询的语义：当表 hotel 中的元组存在且满足谓词 $star < 4$时，Q_1 中的相应输出元组必须存在，并且它是第一个子查询的输出。第二行以相同方式编码第二个子查询的语义。同样，Q_2 的语义被编码为 F_2。

$$\begin{aligned} F_2 \coloneqq\ & (del_3 \leftrightarrow (del_0 \vee \neg(star_0 < 4))) \\ & \wedge (id_3 = id_0) \wedge (star_3 = star_0) \wedge (price_3 = price_0) \end{aligned}$$

最后，两个列表 l_1 和 l_2 的相等性被编码为 F_3：

$$\begin{aligned} F_3 \coloneqq\ & \Big(\sum_{t_i \in l_1} int(t_i.del) = \sum_{t_j \in l_2} int(t_j.del)\Big) \\ & \wedge \bigwedge_{t \in l_1} \Big(\sum_{t_i \in l_1} int((t_i = t) \wedge \neg t_i.del \wedge \neg t.del) \\ & \qquad = \sum_{t_j \in l_2} int((t_j = t) \wedge \neg t_j.del \wedge \neg t.del)\Big) \end{aligned}$$

为简单起见，我们用 $int(p)$ 表示 $ite(p, 1, 0)$。如果公式 p为真，则 $int(p)$ 返回 1；否则，返回 0。项 $t.a$表示符号元组 t 的符号属性。F_3由两部分组成。第一部分 $\sum_{t_i \in l_1} int(t_i.del) = \sum_{t_j \in l_2} int(t_j.del)$ 表示查询 Q_1 和 Q_2 的查询结果应具有相同数量的元组。第二部分表示，对于列表 l_1中的每个元组 t，它在查询 Q_1 和 Q_2 的查询结果中的多重性应相同。t在 Q_1 结果中的多重性通过 $\sum_{t_i \in l_1} int((t_i = t) \wedge \neg t_i.del \wedge \neg t.del)$ 计算。t 在 Q_2 结果中的多重性也以类似方式表示。

最后一步中，基于列表的验证器会调用 SMT 求解器，检查 $F_1 \wedge F_2 \wedge \neg F_3$ 的可满足性。如果公式可满足，则这两个查询并不等价，求得的模型可作为反例。即使对于这个简单示例，界限为 1 且只有四个 SQL 操作，生成的公式也包含 179 个 AST 节点，其中包括 78 个变量和 101 个运算符。在现实

场景中，查询及其对应公式的复杂度都会大幅增加。

3 SQLUTION 概述

我们开发了一种新的完备验证器 SQLution，克服了现有基于列表的方法的局限。与以往方法不同，SQLution 采用一种基于 U-表达式 [20], 的新颖编码，从而大幅简化了公式。由于此前不存在针对 U-表达式的完备验证算法，SQLution 为推理生成的公式引入了一种新的判定过程。除了提升验证效率，SQLution 还支持比现有验证器更广泛的 SQL 特性。

U-表达式。 Chu 等人 [20], 提出的 U-表达式，是一种将查询表示为函数 $f(t)$ 的代数表示，该函数将每个元组 t 映射到其在查询结果中的多重性。U-表达式能够表示广泛的 SQL 特性，包括聚合函数、外连接和并集 [20, 27, 54]。它们还支持以往方法未涵盖的特性，例如交集和差集。

U-表达式的主要项/运算符总结见表 1。每个项都会返回一个非负整数值，并通过“×”或“+”连接在 U-表达式中。具体而言，对于每个关系名 R，都有一个预定义函数 $[\![R]\!](t)$，用于返回元组 t 在关系 R中的多重性。对于每个谓词 b，U-表达式 $[b]$ 的值为 1（b 为真）或 0（b 为假）。为简洁起见，若不存在歧义，我们会在 U-表达式中省略 $[\![]\!]$。为说明 U-表达式，我们遵循 SQLSolver 中的算法 [27]对若干核心 SQL 特性进行编码。

- 选择： $Sel_p(Q)$ 表示对带有谓词 p 的子查询 Q 进行选择。假设 Q 的 U-表达式为 f_l。该选择的 U-表达式为

$$f(t) \coloneqq f_l(t) \times [p(t)].$$

  如果 t 满足 p，则保留其多重性，否则将其设为 0。

- 投影： $Proj_a(Q)$ 表示从子查询 Q 的结果中返回属性 a。假设 Q 的 U-表达式为 f_l。该 U-表达式为

$$f(t) \coloneqq \sum_{t_l} f_l(t_l) \times [t = t_l.a].$$

  无界求和 $\sum_{t_l}$ 聚合所有元组的多重性，其中其投影属性 a 与 t 匹配。

- UNION ALL： $UnionAll(Q_l, Q_r)$ 表示子查询 Q_l 和 Q_r 的并集。假设 Q_l 和 Q_r 的 U-表达式分别为 f_l 和 f_r。该并集的 U-表达式为

$$f(t) \coloneqq f_l(t) + f_r(t),$$

  将两个子查询中 t 的多重性相加。

基本思路SQLution。我们完整的 SQL 等价性验证器 SQLution 基于 U-表达式。给定两个查询 Q_1 和 Q_2，SQLution 首先将它们分别转换为 U-表达式 $f_1(t)$ 和 $f_2(t)$。随后，它将公式 $\exists t.f_1(t) \neq f_2(t)$ 编码为一阶逻辑，可由 SMT 求解器自动检查。如果公式可满足，则存在一个元组，其在两个查询结果中的重数不同；SMT 求解器会返回一个模型，可用于构造反例数据库。

动机示例。现在，我们展示 SQLution 如何反驳图1中的查询，并将所得公式与现有研究中的公式进行比较。首先，这些查询的 U-表达式分别为 $f_1(t)$ 和 $f_2(t)$。

$$f_1(t) \coloneqq (hotel(t) \times [t.star < 4]) + (hotel(t) \times [t.price < 500])$$
$$f_2(t) \coloneqq (hotel(t) \times [t.star < 4])$$

具体而言， $f_1(t)$ 是两个子表达式之和，表示每个子查询中 t 的多重性。随后，SQLu- tion 将 $\exists t.f_1(t) \neq f_2(t)$ 转换为一阶逻辑公式。根据酒店表的模式， t 是一个包含三个整数变量的元组（id_1， $star_1$， $price_1$）。所得的一阶逻辑公式为

$$\Big((hotel(id_1, star_1, price_1) \times int(star_1 < 4)) + (hotel(id_1, star_1, price_1) \times int(price_1 < 500))\Big) \neq (hotel(id_1, star_1, price_1) \times int(star_1 < 4))$$

最后，SMT 求解器检查该公式，并报告查询不等价。

在此示例中，SQLution 生成的公式仅包含 26 个抽象语法树节点，由 15 个变量和 11 个运算符组成，明显小于基于列表的验证器生成的 179 节点公式。我们对源自 Spider 的测试用例进行评估，结果显示，与 Polygon 相比，SQLution 将验证时间中位数缩短了 83.1%。

挑战。直接应用 U-表达式来构建完整的验证器是一项并非易事的任务。首先，单独使用 U-表达式无法表示具有有序包语义的查询。其次，将 U-表达式中的无界求和编码为一阶逻辑颇具挑战性，因为一阶逻辑算子无法表示对无界项求和。我们通过有序包编码（第 4 节）和用于无界求和的完备过程（第 5 节）来解决这些问题。

4 排序语义编码

我们提出排序语义编码，以弥合 U-表达式与会对元组排序的 SQL 查询之间的语义差距。SQL 查询的排序语义可能受到 LIMIT、OFFSET 和 ORDER BY 子句的影响。为处理排序语义，我们的方法分为两部分：（1）SQL 查询预处理，删除所有 LIMIT 和 OFFSET 子句；（2）查询等价性编码，处理 ORDER BY 所涉及的排序语义。关于如何正确处理 LIMIT/OFFSET/ORDER BY 的证明，请参见补充材料的第2节 [6]。

4.1 SQL 查询预处理

消除 LIMIT 和 OFFSET 子句的方法是将包含这些子句的 SQL 查询转换为另一个不包含这些子句的新查询，同时保留查询语义。LIMIT 和 OFFSET 子句会选出排名处于特定范围内的元组。不失一般性，假设元组按其属性 a 的值降序排列。我们的关键见解是，可以通过聚合其他具有更大属性 a值的元组的多重性，来计算每个元组的排名。这种计算可以自然地通过聚合

表 1：U-表达式的术语/算子。

术语/算子	概念	示例查询	示例 U-表达式 $f(t)$
$[\![R]\!](t)$	$[\![R]\!](t)$ 是一个返回元组 t的多重性的函数关系 R中的元组。	选择 * 来自 R	$[\![R]\!](t)$
[b]	[b] 如果谓词 b 为，则返回 1 为真。否则 , 返回 0。	查询 * 自 R 其中 R.x < 500	$[\![R]\!](t) \times [t.x < 500]$
$\sum_t e(t)$	$\sum_t e(t)$ 返回 $e(t_i)$ 的总和，适用于每个可能的元组 t_i，其中 e 是一种类型为元组 $\to \mathbb{N}$的 U-表达式。由于无界求和可能出现 t_i，$\sum_t e(t)$ 被称为无界求和。	选择 R.x 自 R	$\sum_{t_1}([t = t_1.x] \times [\![R]\!](t_1))$

-- 规则 1Q_1:R 按 R.attr 降序排列，LIMIT m OFFSET nQ_2: 从（选择 R1.*，sum（条件表达式 R1.attr <R2.attr 则 1 否则 0 结束）作为 s，从 R 作为 R1 全连接 R 作为 R2，按 R1.attr 分组，筛选 s <m+n 且 s >= n）中选择 R1.*，按 R1.attr 降序排列 -- 规则 2Q_3:R 按 R.attr 升序排列，LIMIT m OFFSET nQ_4: 从（选择 R1.*，sum（条件表达式 R1.attr >R2.attr 则 1 否则 0 结束）作为 s，从 R 作为 R1 全连接 R 作为 R2，按 R1.attr 分组，筛选 s <m+n 且 s >= n）中选择 R1.*，按 R1.attr 升序排列

图 2：用于移除给定 SQL 查询中 LIMIT 和 OFFSET 的预处理规则。每条规则都将第一个 SQL 查询转换为第二个 SQL 查询；这两个查询等价。

函数“sum”。因此，我们可以通过聚合函数生成 LIMIT 和 OFFSET 子句的结果。尽管我们可以完全移除 LIMIT 和 OFFSET 子句，但值得注意的是，通常无法完全移除 ORDER BY 子句。否则，转换后的查询将只有包语义，与具有顺序语义的原始查询不等价。

图 2 展示了两条预处理规则，旨在一般性地从查询中移除所有 LIMIT 和 OFFSET 子句。第一条规则用于处理降序排列，第二条规则用于处理升序排列。对于第一条规则，原始查询 Q_1 首先按照属性 attr 对 R 的结果中的元组进行降序排序，然后跳过前 n 个元组并返回接下来的 m 个元组。请注意，R 可以是子查询或表。SQL 查询 Q_2 通过使用自连接和聚合函数“sum”实现相同的结果。具体而言，对于 R中的每个元组，聚合函数“sum”会统计 R 中属性 attr 值更大的元组数量。基于这一计数，它会筛除元组，仅保留那些在排序序列中位于位置 $n+1$ 到 $m+n$ 之间的元组。随后，结果按照属性 attr 进行排序。请注意该规则要求属性 attr 具有唯一值才能生效。否则，Q_1 会产生歧义，因为具有相同 attr 值的元组的顺序未被定义。SQLution 不考虑有歧义的查询，因为其语义具有非确定性，难以进行精确定义。升序排列的第二条规则与此对称，它统计的是值较小而非较大的元组数量。NULL 排序也可以采用类似方式处理。

移除所有 LIMIT 和 OFFSET 后，SQLution 还会移除某些子查询（例如 FROM 子查询）中的 ORDER BY。由于根据 SQL 标准和主流数据库手册，SQL 会忽略这类 ORDER BY，因此这不会改变查询结果 [50, 55]。在其他情况下，SQLution 会将 ORDER BY 从子查询提升到最外层，这同样不会改变查询语义。最终，每个查询在最外层至多包含一个 ORDER BY 子句。

4.2 查询等价性编码

给定两个经过预处理的 SQL 查询 Q_1 and Q_2，SQLution 会进一步对它们的等价性进行编码。具体而言，SQLution 会检查 Q_1 and Q_2 在最外层是否包含 ORDER BY。针对以下三种可能情况，SQLution 采用不同的方法将 SQL 查询对编码为公式：（1）两个查询都不包含 ORDER BY；（2）只有一个查询包含 ORDER BY；（3）两个查询都包含 ORDER BY。如果所得公式是可满足的，则这两个 SQL 查询不等价。重要的是，与已有工作一样，[30, 60], SQLution 不依赖在实际数据上执行查询。公式中的所有实体，包括元组和输入表，都是符号化的，因此该检查会考虑所有满足界限和完整性约束的可能输入数据库。如果公式是可满足的，SMT 求解器会返回使查询产生不同结果的具体表内容；这些内容由求解器生成，而不是由 SQLution 提供。

情形 1：两个查询均不包含 ORDER BY。

在这种情形下，两个 SQL 查询采用包语义而非顺序语义。假设 f_1 和 f_2 分别是对这两个 SQL 查询进行建模的 U-表达式。验证所用的公式如下：

$$\exists t. f_1(t) \neq f_2(t)$$

该公式说明存在一个元组 t，使其在 Q_1 和 Q_2 的查询结果中的重数不同。如果该公式被证明

不可满足，则意味着这些 SQL 查询等价。否则，意味着存在一个元组 t，使得 t 在两个查询结果中的重数不同。这两个查询不等价，SQLution 将返回一个反例。SQLution 使用一种新的完备判定过程证明该公式，相关内容将在第 5 节讨论。

案例 2：只有一个查询包含 ORDER BY。

步骤 *1*：SQLution 将两个 SQL 查询 Q_1 和 Q_2 作为输入。假设 Q_1 包含 ORDER BY，而 Q_2 不包含 ORDER BY。f_2 是 Q_2的 U-表达式。SQLution 对以下公式进行推理，以检查是否存在一个在界限范围内的符号输入数据库，使得 Q_2 产生两个不同元组。

$$\exists t_1, t_2.((t_1 \neq t_2) \wedge (f_2(t_1) > 0) \wedge (f_2(t_2) > 0)) \quad (1)$$

如果公式可满足，SMT 求解器会返回具体的表内容，使得 Q_2 产生两个不同元组 t_1 和 t_2。如果 Q_1的结果也包含这两个不同元组 t_1 和 t_2，那么 Q_1的结果中的元组是有序的，而 Q_2的结果中的元组是无序的。这两个查询不等价。如果 Q_1的结果不包含 t_1 或 t_2，这两个查询仍然不等价。

步骤 *2*：如果公式 (1) 不可满足，那么对于界限范围内的每个输入数据库，Q_2 返回的要么是空结果，要么是值相同的元组。例如，如果模式声明了一个主键，且查询指定了该键的确切值，那么该查询在每个有效输入数据库上最多返回一个元组。因此，如果 Q_1 中跟在 ORDER BY 后面的子查询与 Q_2等价，那么 Q_1 和 Q_2 也等价。这是因为，对空结果或值相同的元组进行排序不会改变结果。否则，Q_1 和 Q_2 不等价。基于上述分析，SQLution 从 Q_1中提取跟在 ORDER BY 后面的子查询。假设提取出的子查询为 Q_1'，且 Q_1' 的 U-表达式为 f_1。SQLution 会检查以下公式是否可满足。

$$\exists t.f_1(t) \neq f_2(t)$$

该公式指定存在一个元组 t，使得它在 Q_1' 和 Q_2 的结果中的多重度不同。如果该公式可满足，SQLution 将返回反例。否则，两个查询等价。

情况 3。两个查询都有 ORDER BY。

步骤 *1*：SQLution 首先检查 SQL 查询 Q_1和 Q_2的子查询是否等价。如果不等价，则 Q_1 和 Q_2 不等价。设 Q_1' 和 Q_2' 分别是 Q_1 和 Q_2的子查询。f_1 和 f_2 分别是 Q_1' 和 Q_2'对应的 U-表达式。如果$\exists t.f_1(t) \neq f_2(t)$ 可满足，则存在一个元组 t，使得 Q_1' 和Q_2' 不等价。SQLution 将返回 Q_1' 和 Q_2'的反例，该反例也是原始查询 Q_1 和 Q_2的反例。因为 ORDER BY 只对元组排序，而不会改变元组。

步骤 *2*：如果 Q_1' 和 Q_2'的子查询等价，SQLution 将进一步检查 Q_1和 Q_2 是否以相同顺序对元组进行排序。换句话说，SQLution 需要检查是否存在两个元组 t_1 和 t_2，使得：1) 在 Q_1 的结果中，t_1 排在 t_2 之前；2) 在 Q_2的结果中，t_1 排在 t_2 之后。假设 Q_1 和 Q_2 分别按照属性 a_1（顺序为 dir_1）和属性 a_2（顺序为 dir_2），对其子查询生成的元组进行排序。这里的 dir_1 和 dir_2 可以是升序或降序。

算法1： 判定过程

```
输入： 一个包含一阶逻辑项和 F
  U-表达式的公式。
输出： 该公式的可满足性。
证明(F):
  b ：=GetBoundConfig()
  M ：= 将表转换为元组(F, b)
  F1 ：= 展开无界求和(F, M)
  result ：=SMT 求解(F1)
  如果 result = 不可满足 则
    返回 等价
  else
    返回 result
```

- 如果 dir_1 与 dir_2相同，则用于验证的公式为$\exists t_1,\ t_2.((f_1(t_1) > 0) \wedge (f_1(t_2) > 0) \wedge (t_1.a_1 > t_2.a_1) \wedge (t_1.a_2 < t_2.a_2))$

  这意味着存在两个元组 t_1 和 t_2，满足：1） t_1 出现在两个 SQL 查询（$f_1(t_1) > 0$）的结果中；2） t_2出现在两个 SQL 查询（$f_1(t_2) > 0$）的结果中；3）它们在 Q_1 和 Q_2 的结果中的相对顺序不同（（$t_1.a_1 > t_2.a_1$）∧（$t_1.a_2 < t_2.a_2$））。如果上述公式可满足，则 Q_1 和 Q_2 不等价；否则，它们等价。请注意，由于我们已证明 Q_1 和 Q_2 的子查询等价（即，U-表达式 f_1和 f_2 等价），仅凭（$f_1(t_1) > 0$）∧（$f_1(t_2) > 0$）即可推断 t_1 和 t_2 都出现在 Q_1 和Q_2的结果中。

- 如果 dir_1 与 dir_2不同，验证公式为$\exists t_1,\ t_2.((f_1(t_1) > 0) \wedge (f_1(t_2) > 0) \wedge (t_1.a_1 > t_2.a_1) \wedge (t_1.a_2 > t_2.a_2))$

  上述公式的含义与前一个公式相似，除了 $t_1.a_2 > t_2.a_2$。这是因为现在一个查询按升序排列元组，另一个则按降序排列。

5 无界求和展开

本节介绍如何在不损失完备性的情况下，将含有无界求和的 U-表达式转换为纯一阶逻辑公式。我们的基本思路是将有界验证 [12, 35, 38, 47, 49] 整合到 U-表达式中。我们提出了一些规则，将无界求和展开为带有特定符号元组的有界求和。

5.1 基本思路

处理无界求和的挑战源于每个表中不同元组的数量未知。无界求和需要枚举所有可能的元组，因为我们不知道哪些元组会使求和体不为零。然而，我们观察到，如果已知每个表中不同元组的数量，那么使求和体非零的元组数量就有界，因此无界求和可以展开为有界求和。

基于这一洞见，我们提出无界求和扩展，将 U-表达式与有界验证相结合，以更好地建模 U-表达式的求和语义，并增强 SQL 等价性完备验证的能力。算法1展示了我们方法的整体流程。基本

创建表 stu(id 整数, 年龄 整数);Q_1: 查询 id 来自 stu 其中 年龄 > 18Q_2: 查询 id 来自 stu 其中 年龄 + 4 > 22

它们对应的 U-表达式为:

$$f_1(t) \coloneqq \sum_{t_1}([t = t_1.id] \times stu(t_1) \times [t_1.age > 18])$$
$$f_2(t) \coloneqq \sum_{t_2}([t = t_2.id] \times stu(t_2) \times [t_2.age + 4 > 22])$$

图 3：两个等价 SQL 查询及其 U-表达式示例。

其基本思路是将每个表中不同元组的最大数量设为界限，并在该界限下进行验证。界限的值可以通过调用 *GetBoundConfig*（第 4 行）从配置文件中读取。工作流程包括以下步骤。首先，我们根据界限 b构造映射 M，将表名映射到不同符号元组的列表。其次，基于 M，我们将公式 F中的所有无界求和替换为有界求和（详见第 5.2 节）。最后，我们使用现成的 SMT 求解器检查所得纯一阶逻辑公式的可满足性。如果验证条件在某个界限下可满足，则这两个 SQL 查询不等价。SMT 求解器会提供一个反例。否则，用户可以逐步增大界限并重新运行验证器，直到找到反例或验证超时。为推理完整性约束下 U-表达式的等价性，SQLution 采用了与先前工作 [30]类似的方法。假设不含完整性约束的查询等价性编码为 F。首先，将完整性约束编码为不含量词的一阶逻辑公式 P。然后，SQLution 使用算法1检查 $P \wedge F$的可满足性。由于完整性约束的编码方式与先前工作 [30, 60], 类似，此处省略详细信息。

请注意，尽管以往关于 SQL 查询验证的工作采用了有界验证， [30, 60], 但它们的界限与我们的含义不同。在以往的工作中，界限表示整张表的大小，而我们的界限针对的是不同元组的数量，而不是所有元组的数量。例如，如果将界限设为一，现有工作会假设每张表最多包含一个元组，而我们假设每张表最多包含一个不同元组，但可以包含多个具有相同值的元组。因此，在界限相同的情况下，SQLution可能找到现有工作无法找到的反例。例如，考虑 SQL 查询“select a from r”和“select distinct a from r”。当界限为一时，现有验证器无法找到反例，因为它们假设表 r 最多包含一个元组。这两个查询都会返回该元组。然而，在相同的界限下，SQLution可以找到该反例，即一张 r 由两个具有相同值的元组组成的表。此时，“select a from r”将返回两个元组，而“select distinct a from r”只会返回一个元组，因为“distinct”会移除重复元组。因此，SQLution 可以证明这些查询并不等价。

通过确定每个表中的不同元组数量，我们的方法能够准确地（即，不进行近似 [39]）表示可表示为一阶逻辑的项所构成的无界求和。这意味着该方法保留了验证完备性。我们的方法会迭代执行以下三个步骤，直到所有无界求和都扩展为有界求和。首先，算法找到最内层的无界求和。若 $e(t)$ 中不含任何求和，则求和 $\sum_t e(t)$ 为最内层求和。其次，算法将通过递归分析求和体来构造一个有界求和域。最后，假设 $\sum_t e(t)$ 的求和域为 $\{t_1、\ldots\ldots、t_n\}$。原始 U-表达式中的求和 $\sum_t e(t)$ 将扩展为 $e(t_1) + \ldots + e(t_n)$。

对于每个无界求和 $\sum_t e(t)$，直观的做法是将求和域设为由新的符号元组组成的有界集合。然而，这种直观方法存在两个问题。首先，很难确定求和域的大小。域大小应大于能够使求和体非零的不同元组数量，否则，原始求和将无法与有界求和等价。然而，域过大可能会包含不必要的元组，导致公式过长、验证速度变慢。其次，上述方法需要引入一个带量词的约束，用来说明域外的任何元组都会使求和体为零。具体而言，对于无界求和 $\sum_t e(t)$ 和求和域 $\{t_1、\ldots\ldots、t_n\}$，该约束为 $\forall t.((t \neq t_1 \wedge \ldots \wedge t \neq t_n) \rightarrow (e(t) = 0))$。众所周知，SMT 求解器很难处理量词。然而，省略量词会损失完备性，这意味着不可满足的公式可能被错误地判定为可满足。例如，图 3 展示了两个等价查询及其 U-表达式 f_1 和 f_2。要证明它们等价，我们需要证明 $\exists t.f_1(t) \neq f_2(t)$ 不可满足。假设 f_1 中 $\sum_{t_1}$ 的求和域为 $\{a_1、a_2、\ldots\ldots、a_m\}$，而 f_1 中 $\sum_{t_2}$ 的求和域为 $\{b_1、b_2、\ldots\ldots、b_n\}$。那么，公式如下。

$$\exists t. \sum_{i=1}^{m}([t = a_i.id] \times stu(a_i) \times [a_i.age > 18])$$
$$\neq \sum_{i=1}^{n}([t = b_i.id] \times stu(b_i) \times [b_i.age + 4 > 22])$$

原公式不可满足，而上述公式是可满足的，因为当每个 $students(a_i)$均为零且每个 $students(b_i)$均为一时，后一个公式成立。

5.2 有界求和域

为了解决这一难题，基本思路是精确找出每个能使求和体非零的元组，而不是直接使用新的符号元组。在深入细节之前，我们先来看图 3 中的例子。假设界限为 1，这意味着表中至多有一个不同的元组。我们用 x 表示该元组。于是，很明显， f_1 中 $\sum_{t_1}$ 的求和域为$\{x\}$。因为只有当 t_1等于 x时， $stu(t_1)$才可能非零，这意味着只有当 t_1 等于 x时，求和体才可能非零。出于同样的原因， f_2 中 $\sum_{t_2}$ 的求和域也为$\{x\}$。于是， $\exists t.f_1(t) \neq f_2(t)$将变为以下公式。

$$\exists t.([t = x.id] \times stu(x) \times [x.age > 18])$$
$$\neq ([t = x.id] \times stu(x) \times [x.age + 4 > 22])$$

表 2：构造有界求和域 $dom(e,t,M)$ 的规则。该域是一个有限集合，覆盖所有能够使 e 非零的元组 t 的取值。M 是从表名到该表中符号元组集合的映射，在算法1第 5 行创建。

e	$dom(e,\ t,\ M)$
$r(t)$	$M(r)$
$[t = t']$	$\{t'\}$
$[IsNull(t)]$	$\{NULL\}$
$\|\|e_1\|\|$	$dom(e_1,\ t,\ M)$
$e_1 \times \ldots \times e_i \times \ldots \times e_n$，其中 $dom(e_i,\ t,\ M) \neq \emptyset$	$dom(e_i,\ t,\ M)$
$[t.a_1 = v_1] \times \ldots \times [t.a_n = v_n] \times e'$，其中 $t = (a_1,\ \ldots\ldots,\ a_n)$ 且 $notuse(v_1,\ t) \wedge \ldots \wedge notuse(v_n,\ t)$	$\{(v_1, \ldots, v_n)\}$
$e_1 + e_2$，其中 $dom(e_1,t,M) \neq \emptyset$ 且 $dom(e_2,\ t,\ M) \neq \emptyset$	$dom(e_1,t,M) \cup dom(e_2,\ t,\ M)$
$\sum_{t_1} e_1(t_1)$，其中 $dom(e_1,t_1,M) = S$ 且 $S \neq \emptyset$	$dom(\sum_{v \in S} e_1(v)\ t, M)$
其他	$\emptyset$

该公式不可满足，因为 $[x.age > 18]$ 等价于 $[x.age + 4 > 22]$，这意味着这两个 U-表达式等价。

基于上述观察，我们提出了系统地构造有界求和域的规则，如表 2 所示。给定无界求和 $\sum_t e(t)$，我们的规则会构造一个有界元组集合，记为 $dom(e,t,M)$。M是从表名到符号元组集合的映射，该映射在算法 1 的第 5 行创建。每个表 r 仅包含 $M(r)$ 中的元组。$dom(e,t,M)$ 的作用是返回一个有限集合，该集合覆盖 t 的所有值，使得 $e(t) \neq 0$。该集合可以包含满足 $e(t) = 0$ 的 t，因为这不会破坏 $\sum_t e(t) = \sum_{v \in dom(e,t,M)} e(v)$。如果 $e(t)$ 始终为零，或者存在无穷多个 t，使得 $e(t) \neq 0$，那么 $dom(e,t,M)$ 会返回 $\emptyset$。这些规则涵盖了 U-表达式 的所有可能结构，且不会创建任何新的符号变量。下面将详细说明：

- *DOM-Rel*：假设 e 的形式为 $r(t)$，其中 r 是一个未解释函数，返回 t 在表 r中的多重性。当且仅当 t 是表 r中的一个元组时，$r(t)$才非零。因此，$dom(e,\ t,\ M)$ 等于 $M(r)$。
- *DOM-Eq*：假设 e的形式为 $[t = t']$，，其中 t'是另一个符号元组。当且仅当谓词 $t = t'$为真时，它才非零。因此，$dom(e,\ t,\ M)$ 等于 $\{t'\}$。
- *DOM-Null*：假设 e的形式为 $[IsNull(t)]$，，其中谓词 $IsNull$表示 t为 NULL。当且仅当 $IsNull(t)$成立时，它才非零。因此，$dom(e,\ t,\ M)$ 等于 $\{NULL\}$。
- *DOM-Squash*：假设 e的形式为 $\|e_1\|$，其中 e_1是另一个 U-表达式。当且仅当 e_1非零时，它才非零。因此，$dom(e,\ t,\ M)$ 等价于 $dom(e_1,\ t,\ M)$。例如，$dom(\|stu(t)\|,\ t,\ M)$ 等价于 $dom(stu(t)\ t,\ M)$，也就是 $M(stu)$。这意味着，只有表 stu中的元组才能使 $\|stu(t)\|$非零。
- *DOM-Mul1*：假设 e是 U-表达式 e_1、……、 e_n 的乘积。当且仅当每个 $e_i \in \{e_1$、……、 $e_n\}$ 都不为零时，e才不为零。不为零。因此，如果 $\exists e_i.(e = e_1 \times \ldots \times e_i \times \ldots \times e_n) \wedge (dom(e_i,t,M) \neq \emptyset)$，那么 $dom(e,t,M) = dom(e_i,t,M)$。SQ-Lution 会依次检查每个 e_i，并选取第一个满足 $dom(e_i,t,M) \neq \emptyset$ 的 e_i。请注意，尽管 $dom(e,t,M)$ 可能包含一些能使其他某个 $e_j (i \neq j)$ 变为零的元组，但这不会破坏完备性，因为该规则不会遗漏任何能使 e 非零的元组。例如，$dom(stu(t_1) \times [t_1.age > 18], t_1, M)$ 等价于 $dom(stu(t_1)\ t_1, M)$，而后者等于 $M(stu)$。这意味着，只有表 stu 中的元组才能使 $stu(t_1) \times [t_1.age > 18]$ 非零。
- DOM-Mul2：假设元组 t是符号化属性 $a_1 \ldots a_m$ e $[t.a_1 = v_1] \times \ldots \times$, , 的 $[t.a_n = v_n] \times e$ $t.a_i$ a_i 列表，且形式为 ′。具体而言，是元组 t的属性。 $e' v_1, \ldots,$ v_n均为 U-表达式。该规则返回 $(v_1, \ldots, v_n)$，因为只有元组 $(v_1, \ldots, v_n)$ 能使 $e \neq 0$成立。注意，由于 U-表达式 e展开后不应包含t，因此构造的求和域也不应包含 t中的任何属性。因此，该规则还要求每个 $v_i (1 \leq i \leq n)$ 都不得使用 t中的属性，这表示为 $notuse(v_1,t) \wedge \ldots \wedge notuse(v_n,t)$。例如，$dom([t_1.id = 3] \times [t_1.age = 18], t_1, M)$ 等价于 $\{(3, 18)\}$。这意味着只有元组 $(3, 18)$ 能使 $[t_1.id = 3] \times [t_1.age = 18]$ 变为非零。
- DOM-Add：假设 e 的形式为 $e_1 + e_2$，其中 e_1 和 e_2 是另外两个 U-表达式。当且仅当 e_1 非零或 e_2 非零时，它才非零。因此，$dom(e,t,M)$ 等价于 $dom(e_1,t,M) \cup dom(e_2,t,M)$。例如，$dom([t_1 = t] + [IsNull(t_1)], t_1, M)$ 等价于 $\{t, NULL\}$，这意味着 t 和 $NULL$ 都能使 $[t_1 = t] + [IsNull(t_1)]$ 非零。请注意，它要求 ($dom\ e_1\ t\ M \neq \emptyset \wedge dom\ e_2\ t\ M \neq$ (, ,)) ($\emptyset$ $dom\ e_1\ t\ M \neq \emptyset \vee dom\ e_2\ t\ M \neq \emptyset$ (, ,))，而不是 ((, ,)) ((, ,))。例如，$dom(1 + [t = 0], t, M)$ 应为 $\emptyset$，这意味着无论 t 如何，$1 + [t = 0]$ 始终非零。如果该规则使用 $\vee$ 而不是 $\wedge$，就会错误地返回 0，进而使 $dom((1 + [t = 0]) \times r(t)\ t, M)$ 等于 0，而不是 $M(r)$。
- DOM-求和：假设 e 的形式为无界求和 $\sum_{t_1} e_1\ t_1$ $S = dom\ e_1\ t_1\ M$ $dom\ e\ t\ M$ =( ) 和 (, ,)。那么，(, ,) $(\sum_{v \in S} e(),)$。这意味着 SQLution 会递归处理最内层的无界求和。例如，假设 $M\ stu$ = x_1 $dom \sum_{t_1} stu\ t_1 \times [t =$ $t_1.id]\ t\ M$ $stu\ t_1 \times [t = t_1.id]$ ( ) { }，并考虑 ((( )) ,)。对于求和体 ( )，DOM-Rel 推断出 $dom(stu(t_1)$ $t_1, M) = M(stu)$，即 { x_1 1 $dom\ stu\ t_1 \times [t =$ }。接着，$t_1.id]\ t_1\ M$ x_1 DOM-Mul 推断出 (( ) , ,) 也为 { }。因此，DOM-求和将无界求和重写为 $stu(x_1) \times [t = x_1.id]$，之后其余规则推断出 $dom(stu(x_1) \times [t = x_1.id], t, M)$ 为 $\{x_1.id\}$。这意味着 $\sum_{t_1}(stu(t_1) \times [t = t_1.id])$ 仅当 t等于 $x_1.id$ 时才非零。
- *DOM-*其他：如果 e不符合上述任何形式，例如常量或取反，则 $dom(e,\ t,\ M)$ 为 $\emptyset$。例如，$dom(1,\ t,\ M) = \emptyset$，因为对于 t 的所有可能取值，常量 1 都非零。

为进一步说明这些规则，下面再给出两个更复杂的示例。

示例 1 $\sum_{t_1}[t = t_1.id] \times$。考虑图中 U-表达式的无界求和 (( ))。假设界限为 2，且 $M(stu)$ 为 $\{x_1, x_2\}$，其中 x_1 和 x_2 是两个不同的符号元组。根据 *DOM-Rel*，$dom(stu(t_1)\ t_1, M)$ 为 $M\ stu\ 1\ dom\ t_1 \times ()$。进一步应用 *DOM-Mul* 后，$([t = t_1.id] \times stu(\ ),)$ 也为 ( )。然后，( ) 被转换为 $\sum_{t_1 \in \{x_1,x_2\}}([t = t_1.id] \times stu(t_1) \times [t_1.age > 18])$，这是一个有界求和。类似地，$\sum_{t_2 \in\ x_1\ x_2}\ [t = t_2.id] \times ()$ 被转换为 $\{,\}(\ (\ )\ )$。最后，SQLution 将检查以下公式的可满足性：

$$\sum_{t_1 \in \{x_1,x_2\}} ([t = t_1.id] \times stu(t_1) \times [t_1.age > 18]) \neq \sum_{t_2 \in \{x_1,x_2\}} ([t = t_2.id] \times stu(t_2) \times [t_2.age + 4 > 22])$$

我们还添加 $x_1 \neq x_2$，以指定它们是集合 $M(stu)$ 中的不同元组。显然，该公式不可满足，这意味着两个查询等价。

示例 2。考虑查询“select id from (select ∗from stu union all select ∗ from stu)”，它使用与图 3 相同的模式。这个查询仅用于简单说明。U-表达式为 $[t = t_1.id]$)。假设界限为 ，且 ( ) 为 { , }。应用 DOM-Rel 后，$dom(stu(t_1)\ t_1, M)$ 为 $M(stu)$。接着，DOM-Add 推断出 $dom(stu(t_1) + stu(t_1)\ t_1, M)$ 为 $M(stu) \cup M(stu)$，即 $M(stu)$。由于 $dom(stu(t_1) + stu(t_1)\ t_1, M)$ 不为 ∅，DOM-Mul1 进一步推断出 $stu\ t_1\ t_1\ M$ $\times [t = t_1.id],,)$ 为 (( ) ( ) ,)，这也为 ( )。最后，无界求和变为有界求和 $\sum_{t_1 \in\ x_1\ x_2}\ stu\ t_1 + stu\ t_1 \times \{\ ,\}((\ (\ )\ (\ ))\ )$。

5.3 理论分析

对于给定 SQL 查询的 U-表达式中的任意无界求和，表 2 中的规则始终可以推断出一个非空且有界的集合，从而有助于将其转换为等价的有界求和。需要注意的是，尽管很容易构造出一个求和 $\sum_t e(t)$，使得 $dom(e, t, M)$ 是 ∅，但有效 SQL 查询的 U-表达式中不会存在此类求和。下面给出形式化上述陈述的定理。由于篇幅限制，我们省略证明，证明见补充材料的第 3 节 [6]。

定理 5.1。对于 *SQL* 查询的 *U*-表达式中的任意无界求和 $\sum_t e(t)$ 以及映射 M，如果表 *2* 中的规则生成集合 $S := dom(e, t, M)$，则 S 是有界的。

定理 5.2。对于任意 *SQL* 查询 Q 的 *U*-表达式、一个界限 b 和一个映射 M，以下性质成立：

(1) 对于 *U*-表达式中的任意无界求和 $\sum_t e(t)$，表 *2* 中的规则将生成集合 $S := dom(e, t, M)$，使得 $S \neq \emptyset$ 且 $\sum_t e(t) = \sum_{v \in S} e(v)$。

(2) 如果通过表 *2* 中的规则将所有无界求和替换为有界求和（即将上文中的 $\sum_t e(t)$ 替换为 $\sum_{v \in S} e(v)$）后，会生成新的 *U*-表达式 $f'(t)$，那么 $dom(f', t, M) \neq \emptyset$ 且 $\forall v.((f'(v) \neq 0) \rightarrow (v \in dom(f', t, M)))$。

定理 5.3。对于 *SQL* 查询的任意 *U*-表达式 $f(t)$、一个界限 b和一个映射 M，如果将 $f(t)$ 中的无界求和依据规则替换为有界求和后变为 $f'()$，则 $f(t) = f\ t\ '()$。

6 可靠性与完备性

本节给出了关于我们验证方法的可靠性和完备性的一些定理。证明见补充材料 [6] 的第 4 节。

定理 6.1（完备性保证）。对于任意查询Q_1、Q_2、完整性约束 *IC* 和界限 b，如果 *SQLution* 找到了反例，那么 $Q_1 \neq Q_2$。

定理 6.2。存在非等价查询 Q_1、Q_2、完整性约束 *IC* 和界限 b，使得 *SQLution* 无法找到反例。

定理 6.3。对于任意查询 Q_1、Q_2、完整性约束 *IC* 和界限 b，如果存在数据库数据 *DB*，使得 *1*）*DB* 中每个表最多有 b 个不同的行，且 *2*）$Q_1(DB) \neq Q_2(DB)$，那么 *SQLution* 将返回一个反例。我们假设 *SQL* 查询不包含 *SQLution* 不支持的 *SQL* 特性，并且 *SMT* 求解器能够求解生成的 *SMT* 公式。

定理 6.4。*(1)* 对于任意查询 Q_1 和 Q_2、完整性约束 *IC* 以及界限 b，如果 *Polygon* 找到了反例，*SQLution* 就能找到反例。我们假设 *SMT* 求解器能够成功求解生成的公式。

(2) 存在查询 Q_1 和 Q_2、完整性约束 *IC* 以及界限 b，使得 *SQLution* 能找到反例，而 *Polygon* 无法找到。

7 实现

SQLution 复用了 SQL-Solver [27] 的 SQL 解析器和 U-表达式编码器。不过，SQLution 增加了有序包语义和无界求和展开模块。最后，SQLution 将有界 U-表达式编码为纯一阶逻辑公式，并使用 Z3 [25] 对其进行检查。若公式可满足，则可得到一个能区分这两个查询的反例；否则，这两个查询等价。

8 评估

在评估中，我们回答以下问题：1）SQLution 是否比先前工作具有更强的验证能力？2）SQLution 的验证速度是否比先前工作更快？3）SQLution 具备这些优势的主要原因是什么？

8.1 实验设置

基线。为回答上述问题，我们将 SQLution 与 VeriEQL [30] 和 Polygon [60], 进行比较，它们是目前用于 SQL 等价性的最先进完备验证器。评估不会将 SQLution 与可靠验证器进行比较，因为它们并不完备，因此无法作为生成反例的有效基线。例如，近期的可靠验证器 SQL-Solver 无法为任何查询对生成反例，无论这些查询是否等价。为确保比较具有可比性，我们不将 SQLution 与用于 SQL 等价性测试的测试工具进行比较。需要注意的是，先前的评估曾将我们的基线 VeriEQL 和 Polygon 与各种测试工具 [16, 18, 19, 23] 进行比较。结果表明，这些验证器

相比测试工具，具有更强的识别非等价查询对的能力。因此，我们不再重复与测试工具的比较，而是直接将 SQLution 与 VeriEQL 和 Polygon 进行比较。此外，测试工具无法提供完备验证器所具备的理论保证。

基准测试。为评估各方法的验证能力和效率，我们使用了三个基准测试。第一个基准测试包含 997 对源自 Spider [58], 的 SQL 查询。Spider 是文本转 SQL 任务中一个广受欢迎的基准测试。Spider 中的每个测试用例都要求大语言模型将关于某个数据库的自然语言问题转换为 SQL 查询。由于 Spider 被广泛用于评估基于大语言模型的文本转 SQL 模型，因此它包含了具有现实代表性的 SQL 查询，并覆盖了常用的 SQL 特性。我们使用了 DAIL-SQL [29], 的转换结果；它是 Spider 排行榜 [56] 上排名第一的模型。随后，我们将生成的查询与 Spider 提供的对应正确答案配对，构成等价性验证的测试用例。该基准测试包含等价和不等价的 SQL 查询对。为检查每个测试用例中的两个查询是否确实等价或不等价，我们结合使用可靠验证器（例如，SQLSolver [27]）和完备验证器（例如，Polygon）。如果完备验证器判定某对 SQL 查询不等价，或可靠验证器判定其等价，我们便直接采用验证结果。否则，我们会人工检查这些查询。在原始的 1,034 对查询中，726 对等价，271 对不等价。其余 37 对因第 4.1 节所述原因存在歧义，因此我们将其舍弃。

第二个基准测试包含 127 个查询对，这些查询对源自 SQLSolver 使用的 Spark SQL 基准测试 [27]。其中 127 个案例中有 17 个使用了 SQLution 不支持的 SQL 特性，例如窗口函数。原始基准测试包含从 Spark SQL 优化器重写中提取的等价查询对。为了评估反例生成，我们使用大语言模型对每个查询对中的一个查询进行变异，使该查询对变得不等价。我们通过为每个查询对找到一个反例来确认其不等价性。变异后，包含 SQLution 不支持的 SQL 特性的案例数量从 14 个增加到 17 个。这一增长表明，变异并非只构造有利案例，且该基准测试反映了 SQLution 的局限性。

为进一步评估在更复杂验证任务中的验证效率，我们采用了另一个基准测试，其中包含 1,000 个源自 Cubes 的测试用例，该基准测试由 Polygon 构建 [60]。我们没有考虑 VeriEQL 使用的基准测试（例如 Calcite），因为 Calcite 基准测试中的大多数用例都是等价的 [30]。VeriEQL 在该基准测试中仅找到 4 对非等价查询。因此，Calcite 不适合用于体现更快找到反例的优势。与 Spider 和 Spark SQL 基准测试不同，Cubes 中的每个测试用例包含 100 条 SQL 查询，而不是两条。对于每个测试用例，验证器需要将 100 条 SQL 查询的语义编码为一个公式，并找到一个反例，将这些查询划分为不同的等价类。这意味着，在反例上运行这 100 条 SQL 查询时，一些查询会产生相同的结果，而另一些 SQL 查询则会产生不同的结果。该基准测试表明，除了验证两条 SQL 查询是否等价这一传统任务之外，SQLution 还能够对多条 SQL 查询进行推理。Polygon 还构建了另一个

表 3：SQLution、VeriEQL 和 Polygon 在 Spider 与 Spark SQL 基准测试上的比较。该表展示了在每个界限下被证明为不等价的测试用例总数。

基准测试	验证器	界限			
		1	2	3	4
Spider	SQLution	191	257	265	269
	VeriEQL	70	79	81	81
	Polygon	105	138	140	140
Spark SQL	SQLution	63	90	110	110
	VeriEQL	41	54	55	55
	Polygon	45	83	84	84

基准测试，其中每个测试用例包含 50 条 SQL 查询。不过，我们没有在评估中使用它，因为它与上述基准测试类似，而且每个测试用例包含的 SQL 查询更少。请注意，Cubes 基准测试由 Polygon 构建，且仅使用 Polygon 和 SQLution 支持的 SQL 特性。因此，它主要用于评估验证效率。

测试平台。 所有实验均在一台配备 16 个虚拟 CPU 和 32 GiB 内存的机器上运行。该机器采用第三代 Intel Xeon Scalable 处理器，运行 64 位 Ubuntu 24.04 Server。SQLution 和 Polygon 均使用 Z3 4.15.4 [25, 26] 作为底层 SMT 求解器。SQLution 复用 Apache Calcite [11, 15] 的 SQL 解析器来解析 SQL 查询。

8.2 验证能力

识别非等价查询的能力。我们评估 SQLution、VeriEQL 和 Polygon 识别非等价查询的能力。具体而言，我们将界限从 1 调整到 4，并检查每个验证器在相同界限下能够证明多少测试用例为非等价。最大界限为 4，因为继续增大界限时，各验证器识别出的非等价案例数量不再变化。为充分反映这三个验证器的验证能力，我们等待每个验证器正常完成，不发生超时。如表 3 所示，在 Spider 和 Spark SQL 基准测试中，SQLution 证明为非等价的测试用例数量多于 VeriEQL 和 Polygon。在所有界限下，VeriEQL 证明为非等价的案例都少于 Polygon，这证实了 Zhao 等人报告的 Polygon 具有更强验证能力这一结论 [60]。与 Polygon 相比，在 Spider 基准测试中，SQLution 在界限为 1、2、3 和 4 时分别多证明了 86、115、125 和 129 个非等价测试用例。在 Spark SQL 基准测试中，SQLution 在界限为 1、2、3 和 4 时分别多证明了 18、7、26 和 26 个非等价测试用例。Spider 基准测试中的 271 个非等价案例里，SQLution 在界限为 4 时可以证明其中 269 个（99.3%）为非等价，而 Polygon 只能证明 140 个（51.7%）。我们发现，剩余两个案例在界限为 10 时可以被证明为非等价。Spark SQL 基准测试的 127 个案例中，SQLution 在界限为 4 时可以证明其中 110 个（86.6%）为非等价，而 Polygon 只能证明 84 个（66.1%）。由于不支持某些 SQL 特性（例如窗口函数），SQLution 无法证明剩余 17 个非等价案例为非等价。主要

原因是 SQLution 支持的 SQL 特性比 Polygon 更多，如上文所述。大多数案例都源于这一原因。另一个原因是，对于某些非等价 SQL 查询，SQLution 只需比 Polygon 更小的界限就能证明它们为非等价。对于 Cubes 基准测试，由于它只使用 SQLution、VeriEQL 和 Polygon 均支持的 SQL 特性，我们主要用它来评估验证速度，结果见第 8.3 节。

支持 SQL 特性的能力。我们还通过比较 SQLution 和 Polygon 支持的测试用例数量，展示了 SQLution 对更多 SQL 特性的支持；Polygon 的验证能力强于 VeriEQL。SQLution 支持 Polygon 支持的所有 SQL 特性，包括选择、投影、公用表表达式、子查询、聚合函数、并集、UNION ALL、值、内连接、交叉连接、全连接、外连接、ORDER BY、LIMIT、条件表达式和存在。Polygon 不支持、但 SQLution 支持的一些特性包括交集、差集和 OFFSET。具体而言，我们将界限设为 1，并运行每个验证器，直到其正常结束。如果验证器在某个测试用例上正常结束，就意味着它能够支持该用例。在 Spider 和 Spark SQL 基准测试的 1,124 个测试用例中，SQLution 支持 1,107 个，比 Polygon 多支持 337 个。Polygon 不支持的一些用例使用了交集和差集等 SQL 特性。相比之下，SQLution 可以直接用 U-表达式编码这些 SQL 特性。结果展示了 U-表达式对包语义的表达能力。需要注意的是，尽管 U-表达式功能强大，但若没有我们的判定过程（第 5 节），仍无法实现完整的验证器。此外，有 178 个用例使用了 ORDER BY/LIMIT/OFFSET，这些特性无法仅通过 U-表达式支持。SQLution 能够支持它们，得益于有序包语义的编码方法（第 4 节）。Cubes 基准测试由 Polygon 设计，仅使用 Polygon 支持的 SQL 特性。因此，SQLution 支持 Cubes 基准测试中的所有特性。

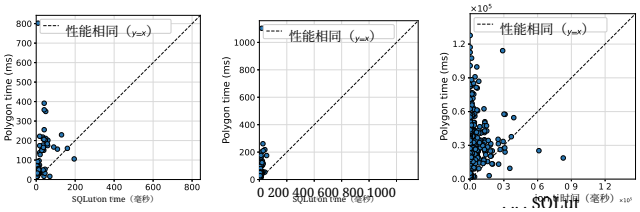
8.3 验证效率

由于 VeriEQL 只能推翻一小部分不等价测试用例，因此我们主要比较 SQLution 和 Polygon 的验证时间，以展示 SQLution 的效率优势。为确保公平性，我们仅考虑 SQLution 和 Polygon 均支持的测试用例。在这些用例中，我们仅考虑非-等价测试用例，因为完备验证器在无法于预定义时间限制内完成时，默认输出“等价”。因此，验证效率仅对非等价案例有意义。对于来自 Spider 和 Spark SQL 的测试用例，我们将界限从 1 逐步提高，直到验证器能够证伪，或界限达到最大值 4；这与第 8.2 节的设置相同。对于来自 Cubes 的测试用例，我们将界限从 1 提高到 3，因为当界限从 2 提高到 3 时，两个验证器都没有发现新的非等价测试用例。

表 4 从 AST 节点数量的角度比较了验证效率和测试用例复杂度。具体而言，对于每个测试用例，我们使用流行的 SQL 解析器 SQL-Glot [24] 解析其所有查询，并统计生成的 AST 中的节点数。SQLution 在 Cubes 基准测试上带来了更大的性能提升。

表 4：基准测试的复杂度以及 SQLution 和 Polygon 的验证时间中位数（毫秒）。“SQL 的 AST 节点数”是一个测试用例中所有查询的 AST 节点总数。为保证公平，我们仅比较两个验证器均支持且包含非等价查询的测试用例的验证时间。

	# SQL 的 AST 节点			SQLution	Polygon
	Avg.	Med.	Max.		
Spider	43	36	158	8.0	47.4
Spark SQL	46	41	98	12.0	53.5
Cubes	4251	4219	7233	95.0	13426.3



(a) Spider (b) Spark SQL (c) Cubes

图 4：SQLution 和 Poly-gon 在 Spider、Spark SQL 和 Cubes 基准测试中的单个测试用例验证时间。每个点表示一个测试用例的验证时间；该测试用例已被两个验证器证明为不等价。虚线表示性能相同。

验证时间中位数从 13.4 秒缩短至仅 95 毫秒，提速超过 100×。这是因为 Cubes 基准测试中的测试用例复杂得多。Cubes 基准测试中的 AST 节点数量中位数是另外两个基准测试的 $> 100\times$。

图 4 比较了 SQLution 和 Polygon 均能证伪的所有案例的验证时间。每个点代表一个测试用例。横轴表示 SQLution 的验证时间，纵轴表示 Polygon 的验证时间。$y = x$ 线表示性能相同。虚 $y = x$ 线以上的点表示 SQLution 比 Polygon 更快的案例。对于 Spider 基准测试中的非等价案例，SQLution 将 92.9% 的测试用例的验证时间缩短了 $\geq 50\%$。43.6% 的测试用例的验证时间可以缩短 $\geq 80\%$。对于 Spark SQL 基准测试，SQLution 将 95.2% 的测试用例的验证时间缩短了 ≥ 50 %，并将 51.2 % 的测试用例的验证时间缩短了 ≥ 80 %。对于 Cubes 基准测试，SQLution 将 94.1% 的测试用例的验证时间缩短了 $\geq 50\%$。82.3% 的测试用例的验证时间可以缩短 $\geq 90\%$。在 Spider、Spark SQL 和 Cubes 上，SQLution 相对于 Polygon 的验证时间中位数缩减比例分别为 78.7%、82.1% 和 97.2%。在 SQLution 和 Polygon 均能证伪的所有测试用例中，SQLution 在 1,124/1,140 个案例（98.6%）中比 Polygon 更快。以下两点成立时，SQLution 会更快：1）SQLution 生成的 SMT 公式比 Polygon 更简单；2）SMT 求解器求解更简单的公式所需时间更短。这两个条件并不总是成立。第一个条件可能

表 5：SQLution 相较于 Polygon 的公式复杂度降幅。对于每个界限值，我们收集满足以下条件的测试用例：(1) 两个验证器都能证伪该界限；(2) 使用更小的界限无法证伪该界限。我们报告复杂度中位数降幅。

界限	Spider			Spark SQL			Cubes		
	大小	# 抽象语法树节点数	# 运算符数	大小	#AST 节点	#运算符	大小	#AST 节点	#运算符
1	-53.0%	-82.8%	-83.1%	-72.0%	-88.9%	-88.9%	-97.0%	-98.7%	-98.4%
2	-87.6%	-96.0%	-95.0%	-82.1%	-93.8%	-92.6%	-98.8%	-99.6%	-99.4%
3	-81.6%	-94.2%	-93.7%	-80.3%	-93.3%	-91.8%	/	/	/
Med.	-59.2%	-85.6%	-86.0%	-76.8%	-91.5%	-90.4%	-97.7%	-99.1%	-98.8%

在 SQL 查询非常简单，且表列数和界限都非常小时失效。第二个条件并不总是成立，因为 SMT 不稳定性 [63]。这意味着，即使两个语义等价的 SMT 查询，有时在同一个 SMT 求解器上的性能也可能大相径庭。

我们进一步分析 SQLution 相较于 Polygon 在效率方面的优势。我们使用三个指标来衡量公式的复杂度，包括大小、AST 节点数量和运算符数量。大小是采用 SMT-LIB 格式的公式字符串长度 [31]。对于每个界限，我们仅考虑满足以下两个条件的非等价测试用例：1）SQLution 和 Polygon 都能在该界限下证伪该案例；2）在更小的界限下，两个验证器都无法证伪该案例。因此，对于 Spider 和 Spark SQL 基准测试，我们不考虑界限 4，因为当界限从 3 增加到 4 时，Polygon 并不会证伪新的非等价案例。对于 Cubes 基准测试，我们不考虑界限 3，因为在该界限下，两个验证器都不会证伪新的非等价案例。如表 5 所示，在三个基准测试中，SQLution 都显著减少了公式大小、AST 节点数量和运算符数量。例如，在 Spark SQL 基准测试中，公式大小、AST 节点数量和运算符数量的中位数降幅分别为 76.8%、91.5% 和 90.4%。Cubes 基准测试中的降幅比例更大，因为 Cubes 基准测试中的测试用例更加复杂。对于 Spider 和 Spark SQL 基准测试，由于不同界限下的测试用例不同，当界限从 2 增加到 3 时，降幅比例会降低。因此，SQLution 更简单的公式解释了其在效率方面的优势。

9 相关工作

SQL 等价性的完备验证。近期工作 [21,30, 53, 60] 在开发 SQL 查询等价性的完备验证器方面取得了显著进展。VeriEQL [30] 通过将每个 SQL 运算符的输入和输出表示为由一阶逻辑公式关联的符号列表，支持 SQL 特性。正如第2节所述，这种基于列表的编码会产生复杂的公式。Polygon [60]通过构造 VeriEQL 公式的欠近似来缩小搜索空间，从而加速验证，但仍保留了基于列表的编码的复杂性。相比之下，SQLution 使用 U-表达式生成简单得多的公式，同时支持更多 SQL 特性。

关于 U-表达式的推理。U-表达式是一种能够编码广泛 SQL 特性的代数结构。以往的工作 [20, 27,54, 55] 提出了多种用于推理其等价性的算法。UDP [20] 使用预定义规则将 U-表达式规范化并比较所得的句法结构，但等价的表达式并不总是具有结构相似的范式。WeTune [55] 则将等价性转换为由 SMT 求解器检查的一阶逻辑公式，并使用手工构造的近似来处理特定形式的无界求和。SQL- Solver [27] 无需预定义规则即可处理任意形式的无界求和。由于所有这些方法都以可靠性为目标，SQLution 需要采用不同的算法来确保完备性。

关于有序包语义的推理。一些现有验证器 [20, 54, 55, 61, 62] 仅支持包语义。近期，一些工作 [27, 30, 60] 提出了支持具有有序包语义的查询的方法。现有方法并不完备或效率低下；SQLution 则完备且更快。

10 局限性

SQLution 存在以下两项局限。首先，SQLution 不支持某些 SQL 特性，例如窗口函数、OVER、PARTITION BY、ROLLUP、CUBE、横向标量子查询和递归 CTE，因为 U-表达式不支持这些特性。经过评估的完备验证器 VeriEQL 和 Polygon [30, 60]也不支持这些特性。其次，如果查询按非唯一属性对元组排序，那么由于排序属性值相同的元组顺序不明确，查询结果将是不确定的。现有的 SQL 等价性验证器 [27, 30, 60] 没有定义此类情况下的查询等价性。因此，SQLution 也无法处理这类含糊的查询。如何将更多 SQL 特性编码为 U-表达式，并处理非确定性 SQL 查询，是未来值得探索的方向。

11 结论

本文介绍了 SQLution，这是一种用于 SQL 查询等价性的完备验证器。我们提出了一种基于 U-表达式的 SQL 查询等价性新编码方法，生成的公式比以往工作简洁得多。我们设计了一种完备的判定过程，用于推理 U-表达式，并可通过无界求和展开处理无界求和。评估结果表明，在支持更多 SQL 特性的同时，SQLution 验证查询的速度远快于当前最先进的 Polygon。

参考文献

[1] 2021。MySQL Bug#104918：派生条件下推重写会忽略用户变量。https://bugs.mysql.com/bug.php?id=104918。[2] 2024。DuckDB Issue # 11294：当子查询不返回任何行时，UNION DISTINCT 和 EXCEPT DISTINCT 会被忽略。https://github.com/duckdb/duckdb/ issues/11294。

[3] 2026。Apache Spark Issue SPARK-58383。https://issues.apache.org/jira/browse/ SPARK-58383。[4] Serge Abiteboul、Richard Hull 和 Victor Vianu。1995。数据库基础。第 8 卷。Addison-Wesley Reading。[5] Rajeev Alur、Rastislav Bodik、Garvit Juniwal、Milo M. K. Martin、Mukund Raghothaman、Sanjit A. Seshia、Rishabh Singh、Armando Solar-Lezama、Emina Torlak 和 Abhishek Udupa。2013。语法引导的综合。载于 *2013* 年计算机辅助设计形式化方法。 1 8 10 1109 2013 6679385 –。[6] 2026 4 https://doi.org/。/FMCAD。。作者。。补充材料。https://anonymous. open.science/r/sqlution-vldb2027-supplementary-material-D826。[7] Qiushi Bai、Sadeem Alsudais 和 Chen Li。2023。QueryBooster：使用中间件服务进行以人为中心的查询重写以提升 SQL 性能。*VLDB*基金会会议录。16，11 36114 79 2023 2911 2924 10 14778 3611497（7 月），–。https://doi.org/。/。[8] Haniel Barbosa、Clark W. Barrett、Martin Brain、Gereon Kremer、Hanna Lachnitt、Makai Mann、Abdalrhman Mohamed、Mudathir Mohamed、Aina Niemetz、Andres Nötzli、Alex Ozdemir、Mathias Preiner、Andrew Reynolds、Ying Sheng、Cesare Tinelli 和 Yoni Zohar。2022。cvc5：一种多用途的工业级 SMT 求解器。载于 *系统构造与分析工具和算法——第 28 届国际会议 TACAS 2022，作为欧洲软件理论与实践联合会议 ETAPS 2022 的一部分举行，德国慕尼黑，2022 年 4 月 2—7 日*，论文集，第 *I* 部分（计算机科学讲义），Dana Fisman 和 Grigore Rosu （编），第 13243 卷。Springer，415–442。https://doi.org/ 10.1007/978-3-030-99524-9 24 –[9] Clark W. Barrett、Christopher L. Conway、Morgan Deters、Liana Hadarean、Dejan Jovanovic、Tim King、Andrew Reynolds 和 Cesare Tinelli。2011。CVC4。载于 *计算机辅助验证——第 23 届国际会议 CAV 2011，美国犹他州 Snowbird，2011 年 7 月 14—20 日*，论文集（计算机科学讲义），Ganesh Gopalakrishnan 和 Shaz Qadeer（编），第 6806 卷。Springer，171–177。https: //doi.org/ 10 1007 978 3 642 22110 1_14 。/ - - - - Pascal Baumann、Rupak Majumdar、Ramanathan S. Thinniyam 和 Georg Zetsche。2021。多线程共享内存程序活性性质的上下文有界验证。*ACM 程序设计语言会议录*。5，POPL，文章 44 2021 31 10 1145 3434325 （1 月），页码。https://doi.org/。/[10] [11] Edmon Begoli 、 Jesús Camacho-Rodríguez 、 Julian Hyde 、 Michael J. Mior 和 Daniel Lemire。2018。Apache Calcite：面向异构数据源的优化查询处理基础框架。载于 *2018* 年国际数据管理会议论文集（美国得克萨斯州休斯敦）（*SIGMOD '18*）。美国计算机协会，纽约，NY，美国，221–230。https: //doi.org/ 10 1145 3183713 3190662 。/。[12] Armin Biere、Alessandro Cimatti、Edmund M. Clarke 和 Yunshan Zhu。1999。不使用二元决策图的符号模型检测。载于 *第 5 届系统构造与分析工具和算法国际会议论文集（TACAS '99*）。Springer-Verlag，德国柏林、海德堡，193–207。[13] Ricardo Brancas、Miguel Terra-Neves、Miguel Ventura、Vasco Manquinho 和 Ruben Martins。2024。迈向可靠的 SQL 综合：基于模糊测试的评估与消歧。载于 *软件工程基础方法：第 27 届国际会议 FASE 2024，作为欧洲软件理论与实践联合会议 ETAPS 2024 的一部分举行，卢森堡市，卢森堡，2024 年 4 月 6—11 日*，论文集（卢森堡市，卢森堡）。Springer-Verlag，德国柏林、海德堡， 232 254 10 1007 978 3 031 57259 3_11 –。[14] https://doi.org/。/ - - - - Ricardo Brancas、Miguel Terra-Neves、Miguel Ventura、Vasco Manquinho 和 Ruben Martins。2025。CUBES：一种使用示例进行 SQL 综合的并行综合器。计算机形式化方法。（2025 年 9 月）。https://doi.org/10.1145/3768578 已接受。[15] Apache Calcite。2025。Apache Calcite 项目。http://calcite.apache.org。[16] J eroen Castelein、Maurício Aniche、Mozhan Soltani、Annibale Panichella 和 Arie van Deursen。2018。面向 SQL 查询的基于搜索的测试数据生成。载于 *第 40 届国际软件工程会议论文集（瑞典哥德堡）（ICSE '18*）。美国计算机协会，纽约，NY，美国， 1220 1230 10 1145 3180155 3180202 –。https://doi.org/。/。Ashok K. Chandra 和 Philip M. Merlin[17] 1977。。关系数据库中合取查询的最优实现。载于 *第九届美国计算机协会计算理论研讨会论文集（美国科罗拉多州博尔德）（STOC '77*）。美国计算机协会，纽约，NY，美国，77–90。https: //doi.org/ 10 1145 800105 803397 。/。Bikash Chandra、Ananyo Banerjee、Udbhas Hazra、Mathew Joseph 和 S. Sudarshan。2019。SQL 查询的自动评分。载于 *2019 IEEE 第 35 届国际数据工程会议（ICDE*）。1630–1633。https://doi.org/10.1109/ICDE。2019.00159[18] [19] Bikash Chandra、Bhupesh Chawda、Biplab Kar、KV Maheshwara Reddy、Shetal Shah 和 S Sudarshan。2015。用于测试和评分 SQL 查询的数据生成。*VLDB*期刊24，6（2015），731–755。[20] Shumo Chu、Brendan Murphy、Jared Roesch、Alvin Cheung 和 Dan Suciu。2018。判定 SQL 查询语义等价的公理基础与算法。*VLDB*基金会会议录。11，11（2018 年 7 月），1482–1495。https://doi.org/10 .14778/3236187.3236200

[21] Shumo Chu、Chenglong Wang、Konstantin Weitz 和 Alvin Cheung。2017。《Cosette：用于 SQL 的自动定理证明器》。载于第八届创新数据系统研究双年会议论文集（美国加利福尼亚州查米纳德）（CIDR '17）。[22] Shumo Chu、Konstantin Weitz、Alvin Cheung 和 Dan Suciu。2017。《HoTTSQL：使用单值化 SQL 语义证明查询重写》。SIGPLAN Not. 52, 6（ 2017 510 524 10 1145 3140587 3062348 年6月），–。[23] 2013 https://doi.org/。/。Fabien Coelho。 。DataFiller：根据数据库模式生成随机数据。https://github.com/memsql/datafiller。[24] SQLGlot 贡献者。2025。SQLGlot：Python SQL 解析器和转译器。https: //github.com/tobymao/sqlglot。[25] Z3 贡献者。2025。z3-4.15.4。https://github.com/Z3Prover/z3。[26] Leonardo De Moura 和 Nikolaj Bjørner。2008。Z3：高效的 SMT 求解器。载于《软件理论与实践》第14届系统构造与分析工具和算法国际会议论文集（匈牙利布达佩斯）（TACAS '08/ETAPS'08）。Springer-Verlag，德国柏林、海德堡，337–340。[27] Haoran Ding、Zhaoguo Wang、Yicun Yang、Dexin Zhang、Zhenglin Xu、Haibo Chen、Ruzica Piskac 和 Jinyang Li。2023。使用线性整数算术证明查询等价性。《美国计算机协会管理数据汇刊》1, 4，文章 227（2023年12月），26页。[28] Rui Dong、Jie Liu、Yuxuan Zhu、Cong Yan、Barzan Mozafari 和 Xinyu Wang。2023。SlabCity：使用程序综合进行全查询优化。《VLDB基金会会刊》16, 11（2023年7月），3151–3164。https://doi.org/10.14778/3611479.3611515 [29] Dawei Gao、Haibin Wang、Yaliang Li、Xiuyu Sun、Yichen Qian、Bolin Ding 和 Jingren Zhou。2024。大语言模型赋能文本到 SQL：基准评测。《VLDB基金会会刊》17, 5（2024年1月），1132–1145。[30] Yang He、Pinhan Zhao、Xinyu Wang 和 Yuepeng Wang。2024。VeriEQL：针对复杂 SQL 查询及完整性约束的有界等价性验证。《美国计算机协会程序设计语言汇刊》8, OOPSLA1，文章132（2024年4月），29页。[31] SMT-LIB 计划。2025。SMT-LIB：理论可满足性库。https://smt-lib.org。[32] Omar Inverso、Ermenegildo Tomasco、Bernd Fischer、Salvatore La Torre 和 Gennaro Parlato。2021。通过惰性顺序化对多线程程序进行有界验证。《美国计算机协会程序设计语言系统汇刊》44, 1，文章1（2021年12月），50页。https://doi.org/10.1145/3478536[33] Zhe Jiang、Zhaoguo Wang、Haoning Lan、Chuzhe Tang、Haoran Ding、Lefeng Wang、Songyun Zou、Zhuoran Wei、Yongcun Liu、Xiang Yu、Yang Ren、Guoliang Li 和 Haibo Chen。2025。GRewriter：在 GaussDB 中通过自动扩展规则集实现实用的查询重写。《VLDB基金会会刊》18, 12（2025年8月），4991–5003。https://doi.org/ 10 14778 3750601 3750622 。/。Michalis Kokologiannakis、Ori Lahav、Konstantinos Sagonas 和 Viktor Vafeiadis。2017。针对 C/C++ 并发的高效无状态模型检查。《美国计算机协会程序设计语言汇刊》2, POPL，文章17（2017年12月），32页。https://doi.org/10.114 5/3158105[35] Michalis Kokologiannakis、Azalea Raad 和 Viktor Vafeiadis。2019。在无状态模型检查中高效处理锁。《美国计算机协会程序设计语言汇刊》3, OOPSLA，文章 173 2019 26 10 1145 3360599 （10月），页数。https://doi.org/。/。Michalis Kokologiannakis、Azalea Raad 和 Viktor Vafeiadis。2019。针对弱一致性库的模型检查。载于第40届美国计算机协会 SIGPLAN 编程语言设计与实现会议论文集（美国亚利桑那州凤凰城）（PLDI 2019）。美国计算机协会，美国纽约州纽约市， 96 110 10 1145 3314221 3314609 –。[36] [37] https://doi.org/。/。Michalis Kokologiannakis 和 Viktor Vafeiadis。2020。HMC：针对硬件内存模型的模型检查。载于第二十五届编程语言与操作系统体系结构支持国际会议论文集（瑞士洛桑）（ASPLOS '20）。美国计算机协会，美国纽约州纽约市， 1157 1171 10 1145 3373376 3378480 –。[38] 2021 https://doi.org/。/。Michalis Kokologiannakis 和 Viktor Vafeiadis。 。GenMC：一种弱内存模型的模型检查器。载于《计算机辅助验证：第33届国际会议，CAV 2021，线上会议，2021年7月20–23日，论文集，第一部分》。Springer-Verlag，德国柏林、海德堡，427–440。https://doi.org/10.1007/978-3-030-81685-8 20[39] –Maxwell Levatich、Nikolaj Bjørner、Ruzica Piskac 和 Sharon Shoham。2020。使用近似求解 LIA*。载于验证、模型检查与抽象解释国际会议。Springer，360–378。[40] Boyan Li、Yuyu Luo、Chengliang Chai、Guoliang Li 和 Nan Tang。2024。自然语言到 SQL 的黎明：我们是否已做好充分准备？《VLDB基金会会刊》17, 11 2024 3318 3331 10 14778 3681954（7月），–。https://doi.org/。/。[41] 3682003 2023

Haoyang Li、Jing Zhang、Cuiping Li 和 Hong Chen。 。RESDSQL：解耦文本到 SQL 的模式链接与骨架解析。载于第三十七届人工智能促进会人工智能会议、第三十五届人工智能创新应用会议和第十三届人工智能教育进展研讨会论文集（AAAI'23/IAAI'23/EAAI'23）。AAAI Press，文章 1466, 9 10 1609 37 11 26535 页。https://doi.org/。/aaai.v i 。Jie Liu 和 Barzan Mozafari[42] 2024。。通过大语言模型进行查询重写。arXiv:arXiv:[43] Jie Liu 和 Barzan Mozafari。2026。GenRewrite：通过大语言模型进行查询重写。《美国计算机协会管理数据汇刊》4, 1，文章70（2026年4月），26页。

https://doi.org/10.1145/3786684[44] Qiuyang Mang、Runyuan He、Suyang Zhong、Xiaoxuan Liu、Huanchen Zhang 和 Alvin Cheung。2025。使用大语言模型自动发现数据库管理系统的测试预言机。arXiv:arXiv:[45] MySQL。2024。行排序。https://dev.mysql.com/doc/refman/9.5/en/sorting-rows.html。[46] Kartik Nagar、Prasita Mukherjee 和 Suresh Jagannathan。2020。复制系统中并发库的语义、规范与有界验证。载于 计算机辅助验证：第*32*届国际会议，*CAV 2020*，美国加利福尼亚州洛杉矶，*2020*年*7*月*21–24*日，论文集，第一部分 （美国加利福尼亚州洛杉矶）。Springer-Verlag，德国柏林、海德堡，251–274。https://doi.org/10.1007/978-3-030-53288-8 13_[47] Jonas Oberhauser、Rafael Lourenco de Lima Chehab、Diogo Behrens、Ming Fu、Antonio Paolillo、Lilith Oberhauser、Koustubha Bhat、Yuzhong Wen、Haibo Chen、Jaeho Kim 和 Viktor Vafeiadis。2021。VSync：针对弱内存模型下同步原语的即点即用验证与优化。载于 第*26*届美国计算机协会编程语言与操作系统体系结构支持国际会议论文集 （线上，美国）（*ASP- LOS '21*）。美国计算机协会，美国纽约州纽约市，530–545。https://doi.org/ 10 1145 3445814 3446748 。/。Mohammadreza Pourreza 和[48] Davood Rafiei。2023。DIN-SQL：通过自校正进行文本到 SQL 的分解式上下文学习。载于 第*37*届神经信息处理系统国际会议论文集 （美国路易斯安那州新奥尔良）（*NIPS '23*1577, 10 ）。Curran Associates Inc.，美国纽约州雷德胡克，文章页数。[49] Tobias Schuele 和 Klaus Schneider。2007。无限状态系统的有界模型检查。*Form. Methods Syst. Des.* 30, 1（2007年2月），51–81。https://doi.org/10.1007/s10703-006-0019-9[50] Snowflake。2026。Snowflake。https://docs.snowflake.com/en/sql-reference/constructs/order-by。[51] Chenglong Wang、Alvin Cheung 和 Rastislav Bodik。2017。根据输入输出示例进行交互式查询综合。载于 *2017*年美国计算机协会国际数据管理会议论文集 （美国伊利诺伊州芝加哥）（*SIGMOD '17*）。美国计算机协会，美国纽约州纽约市，1631–1634。https://doi.org/ 10 1145 3035918 3058738 。/。Chenglong Wang、Alvin Cheung 和 Rastislav Bodik。2017。根据输入输出示例综合高表达力的 SQL 查询。载于 第*38*届美国计算机协会 *SIGPLAN* 编程语言设计与实现会议论文集 （西班牙巴塞罗那）（*PLDI 2017*）。美国计算机协会，美国纽约州纽约市，[52] 452 466 10 1145 3062341 3062365 –。https://doi.org/。/。Chenglong Wang、Alvin Cheung 和 Rastislav Bodik。2018。。加速关系查询的符号推理。美国计算机协会程序设计语言汇刊 2, OOPSLA，文章[53] 157 2018 25 10 1145 3276527 （10月），页数。https://doi.org/。/。Shuxian Wang、Sicheng Pan 和 Alvin Cheung。2024。。QED：强大的 SQL 查询等价性判定器。*VLDB*基金会会刊 17, 11（2024），3602–3614。[54] Zhaoguo Wang、Zhou Zhou、Yicun Yang、Haoran Ding、Gansen Hu、Ding Ding、Chuzhe Tang、Haibo Chen 和 Jinyang Li。2022。WeTune：自动发现并验证查询重写规则。载于 *2022*年美国计算机协会国际数据管理会议论文集 （美国宾夕法尼亚州费城）（*SIG- MOD '22*）。美国计算机协会，美国纽约州纽约市，94–107。https://doi.org/ 10 1145 3514221 3526125 。/。Yale-LILY。。Spider：Yale 语义解析与文本到 SQL 挑战赛。https://yale-lily.github.io/spider。[56] 2018[57] Yicun Yang、Zhaoguo Wang、Yu Xia、Zhuoran Wei、Haoran Ding、Ruzica Piskac、Haibo Chen 和 Jinyang Li。2025。通过执行一致性自动验证并修复文本到 SQL 转换。美国计算机协会管理数据汇刊 3, 3，文章134（2025年6月），28页。https://doi.org/10.1145/3725271[58] Tao Yu、Rui Zhang、Kai Yang、Michihiro Yasunaga、Dongxu Wang、Zifan Li、James Ma、Irene Li、Qingning Yao、Shanelle Roman、Zilin Zhang 和 Dragomir Radev。2018。Spider：用于复杂跨领域语义解析和文本转 SQL 任务的大规模人工标注数据集。载于 *2018*年自然语言处理经验方法会议论文集，Ellen Riloff、David Chiang、Julia Hockenmaier 和 Jun'ichi Tsujii（编）。美国计算语言学协会，比利时布鲁塞尔，3911 186 53 3921 10 1 18 1425 –。https://doi.org/。/v /D -[59] Guoqiang Zhang、Yuanchao Xu、Xipeng Shen 和 Işıl Dillig。2021。通过组合式惰性归纳综合将 UDF 翻译为 SQL。美国计算机协会程序设计语言汇刊 5, OOPSLA，文章112（2021年10月），26页。https://doi.org/10.1145/3485 489[60] Pinhan Zhao、Yuepeng Wang 和 Xinyu Wang。2025。Polygon：使用冲突驱动欠近似搜索对 SQL 进行符号推理。美国计算机协会程序设计语言汇刊 9, PLDI，文章200（2025年6月），26页。[61] Qi Zhou、Joy Arulraj、Shamkant Navathe、William Harris 和 Dong Xu。2019。使用理论可满足性自动验证查询等价性。*VLDB*基金会会刊 12, 11 3 342263 2019 1276 1288 10 14778 3342267（7月），–。https://doi.org/。/。[62] Qi Zhou、Joy Arulraj、Shamkant B. Navathe、William Harris 和 Jinpeng Wu。2022 年。SPES：一种在包语义下证明查询等价性的符号方法。载于*2022 IEEE* 第*38*届国际数据工程会议（*ICDE*）。2735–2748。https://doi.org/10.1109/ICDE53745.2022.00250[63] Yi Zhou、Jay Bosamiya、Yoshiki Takashima、Jessica Li、Marijn Heule 和 Bryan Parno。2023 年。Mariposa：测量自动程序验证中的 SMT 不稳定性。载于*2023* 计算机辅助设计中的形式化方法（*FMCAD*）。178–188。https://doi.org/10.34727/2023/isbn.978-3-85448-060-0_26