合成最小化完整性维护和更新开销的第三范式模式

通过最小键和函数依赖的数量对 3NF 进行参数化

张卓星，新西兰奥克兰大学 赛巴斯蒂安·林克
新西兰奥克兰大学

最先进的关系模式设计会生成无损且保持依赖关系的分解，使其达到第三范式（3NF），并在可能的情况下达到鲍伊-科德范式（BCNF）。特别是，保持依赖关系确保了可以在不连接各个关系模式的情况下维护数据完整性，但可能需要容忍事先无法确定的数据冗余和完整性故障的水平。作为我们的主要贡献，我们通过关系模式所具有的最小键和函数依赖的数量来对其进行参数化。从概念上讲，这些参数在模式设计阶段就量化了维护数据完整性所需的工作量，并使我们能够区分不同的 3NF 模式。从计算角度来看，这些参数使我们能够根据不同的策略优化规范化为 3NF 的过程。从操作角度来看，我们通过实验表明，我们的优化从逻辑层面转化为在维护完整性期间显著减少的更新开销。因此，我们的框架提供了访问参数的途径，这些参数指导逻辑模式设计的计算，从而减少操作开销。

CCS 概念：·信息系统 → 数据库设计与模型；关系数据库模型；完整性检查。

附加关键词和短语：关键；函数依赖；范式；规范化；查询；更新

ACM 参考格式：

张卓星和塞巴斯蒂安·林克。2025 年。《合成最小化完整性维护和更新开销的第三范式模式：通过最小键和函数依赖的数量对 3NF 进行参数化》。《ACM 数据管理杂志》第 3 卷第 3 期（SIGMOD），第 225 篇文章（2025 年 6 月），共 25 页。https://doi.org/10.1145/3725362

1 简介

我们将重新审视关系数据模型中的经典范式，特别是基于函数依赖（FD）的第三范式（3NF）和博伊斯 - 科德范式（BCNF）[6, 11, 18]。这些主题是基础性的，在数据库入门课程中都会讲授[20, 26, 31]。我们不会讨论更高的范式[8, 21, 36]。

可以说，第三范式（3NF）是数据库实践中最流行的范式。虽然 Boyce-Codd 范式（BCNF）保证了任何关系都不会出现冗余的数据值[6]，但将其无损且依赖关系保持地分解为 BCNF 并非总是可行的[5]。相比之下，将其无损且依赖关系保持地分解为 3NF 总是可行的[6, 7, 41]，但可能需要容忍无限制的数据冗余和潜在的完整性故障[6, 16, 23]。若不保持依赖关系，维护数据完整性则被认为代价过高，因为关系模式需要

Authors' Contact Information: Zhuoxing Zhang, zzha969@aucklanduni.ac.nz, University of Auckland, Auckland, New Zealand; Sebastian Link, s.link@auckland.ac.nz, University of Auckland, Auckland, New Zealand.

ACM 2836-6573/2025/6-ART225
https://doi.org/10.1145/3725362

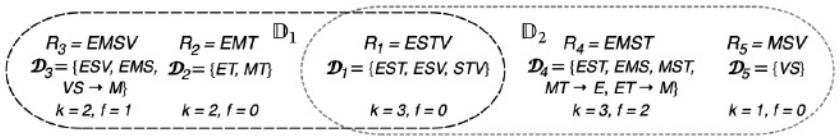

图 1.运行示例的分解 $D_1 = \{R_1,R_2,R_3, \}$ 和 $D_2 = \{R_1,R_4,R_5\}$ 及其参数

在验证 FD 之前必须先进行连接 [7, 23]。进一步推广使用 3NF 是因为它在所有无损且依赖关系保持的分解中，数据冗余的来源最少 [17]。

最先进的规范化算法会计算出一种无损且保持依赖关系的分解，使其达到第三范式（3NF），并且在可能的情况下达到 Boyce-Codd 范式（BCNF）[28, 42]。直观地说，这些算法会检查每一个关键模式（即处于 3NF 但不在 BCNF 的关系模式）是否冗余。如果存在任何非冗余的关键模式，则无法实现无损且保持依赖关系且处于 BCNF 的分解。尽管 3NF 已成为近 50 年来数据库教育和实践中的最基础主题之一，但最先进的算法在处理冗余的关键模式时仍会做出任意选择。以下对此进行了说明。

示例 1.1. 我们将模式 R = {E(vent), M(anager), S(tatus), V(enue), T(ime)} 作为运行示例。它记录了活动的名称、其管理者、状态、举办地点和举办时间。函数依赖集 D 包含以下内容：VSE → T，SET →V，SME →V，VS →M，SME →T，MT →E 和 ET →M。基于任意选择，最先进的规范化方法[28, 42]可能会返回不同的分解，例如图 1 中可视化展示的 （R， D） 的 D1 和 D2。这两种分解都包含具有 3 个最小键的 BCNF 模式 R1，而 D1 还包含具有 2 个最小键的 BCNF 模式 R2，D2 包含仅具有 1 个最小键的 BCNF 模式 R5，同时 D1 包含具有 1 个函数依赖的 3NF 模式 R3，D2 包含具有 2 个函数依赖的 3NF 模式 R4。 □

示例 1.1 展示了当前最先进的技术所面临的挑战。首先，没有系统的方法来区分 3NF 模式之间的优劣。因此，我们提出问题（Q1）：什么样的 3NF 模式更好？在模式设计阶段，目标数据库的未来工作负载尚不可知，3NF 合成的唯一输入是一些模式及其函数依赖集 D。其次，当前的 3NF 条件无法提供足够的信息来区分更好的 3NF 模式和较差的 3NF 模式。这导致了问题（Q2）：我们如何改进现有的第三范式条件以识别更好的 3NF 模式？第三，即使能够区分更好的和较差的 3NF 模式，也还不知道如何优化 3NF 合成。因此，我们提出问题（Q3）：我们可以在何种意义上优化 3NF 合成？最后，我们希望在操作层面看到更好的结果，即最小化完整性维护和更新开销。因此，我们想知道（Q4）：逻辑层面的优化如何在操作层面得以体现？这些问题的答案将丰富关于关系数据库的知识，同时也将推进现代数据模型的发展，例如不完整数据模型[19, 39]、时态数据模型[13]、网络数据模型[2, 9, 40]、不确定数据模型[22]以及图数据模型[1, 32–34]。

我们将按如下方式解决研究问题。我们将使用迈尔关于最小约简覆盖和最优覆盖的开创性概念[25]来解决（Q1）。我们的主要思路是在逻辑层面衡量函数依赖维护的操作工作量，即通过表示函数依赖所需的规模来衡量。我们将函数依赖集 D 分解为它所蕴含的最小键集 K 和非键函数依赖集的最小约简覆盖 F[25]，即左部不是键的函数依赖。因此，K 中元素的数量 k 衡量了转移到最小键的维护工作量，而 F 中元素的数量 f 衡量了维护非键函数依赖的工作量。由此，我们可以基于 k 和 f 对 3NF 模式进行比较。

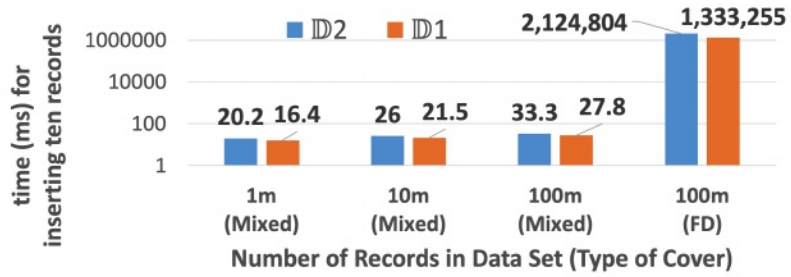

图 2. 在 100 万、1000 万和 1 亿条记录上插入 10 条记录的更新开销（毫秒）：分解 $D_1 = \{R_1,R_2,R_3\}$ 和分解 $D_2 = \{R_1,R_4,R_5\}$

在处理（Q2）时，我们将细化众所周知的 3NF 条件，从而获得参数 k 和 f。因此，我们可以根据所采用的策略来区分 3NF 模式之间的优劣。在我们的示例中，如果策略倾向于选择非主键函数依赖更少的模式（S1），那么 D1 优于 D2。然而，如果策略倾向于选择具有更多最小键的模式（S2），那么 D2 优于 D1。

在最先进的 3NF 合成过程中，我们不再选择任意冗余的关键模式，而是可以选择由我们的策略所偏好的模式。简单来说，我们新的 3NF 合成的输出将针对我们的目标策略进行优化。在我们的示例中，D1 是针对（S1）进行优化的，而 D2 是针对（S2）进行优化的，这说明了我们对（Q3）的处理方式。

在回答（Q4）时，我们注意到非主键的函数依赖（FD）是维护数据完整性最大的瓶颈：其验证速度比最小键的验证速度慢几个数量级。因此，最大的性能提升有望来自加快非主键函数依赖的维护速度。然而，这可以表述为目标策略，即我们示例中的（S1）。正如我们将要展示的，（S1）将优化从逻辑层面转化为操作层面，在此层面中，完整性维护和更新开销被最小化，从而回答了（Q4）。对于我们的示例，图 2 展示了一项研究，该研究测量了示例 1.1 中分解所导致的更新开销。我们分别使用了 100 万、1000 万和 1 亿条合成数据记录作为底层数据集，每条记录都满足给定的函数依赖集，但违反了该集未隐含的每一个函数依赖。因此，这些数据集完美地代表了给定的约束集。然后，这些数据集分别投影到分解 D1 和 D2 的每个元素上。蓝色（橙色）条形图显示了将 10 条记录插入 D1 和 D2 的投影数据集所花费的时间，这衡量了算法输出所使用的维护完整性的努力。报告的时间是 30 次运行的平均值。右侧的两个条形图使用一个 FD 覆盖，该覆盖将所有约束（非主键 FD 和最小键）统一作为 FD 使用触发器来强制执行，而其他地方则使用非主键 FD 和最小键（及其 UNIQUE 索引）的混合覆盖[43, 44]。主要有两个观察结果：（1）混合覆盖使完整性维护的速度比 FD 覆盖快几个数量级。（2）策略（S1）在 D1 上产生的更新开销明显小于策略（S2）在 D2 上产生的更新开销，这在所有数据集大小和两种类型的覆盖中都成立。这为我们的逻辑层归一化策略如何在操作层减少基于策略（S1）的更新开销提供了定量的见解。

我们的贡献可总结如下。

从根本上说，我们引入了量化维护数据完整性所需工作量的参数，从而能够打破 3NF 模式的平局。为此，我们通过将非主键的函数依赖与最小键在最小约简和最优覆盖中分离，对经典的关系规范化进行了参数化处理[25]。

（2）从算法角度来看，我们的框架使我们能够根据参数设定的策略优化无损且保持依赖关系的分解为第三范式。这用策略性选择取代了随意选择，从而改进了现有技术，因为我们能够声明如何打破冗余第三范式模式之间的平局。通过使用真实和合成的函数依赖集进行的实验展示了我们的算法所返回的模式设计在质量上优于以往的工作。

（3）在操作层面，我们评估了优化措施在多大程度上减少了维护更新的开销。我们揭示了哪种策略在最小化更新开销方面效果最佳，展示了相对于之前工作的性能提升，以及逻辑层面的优化在多大程度上能提升完整性。

总体而言，我们建立了首个针对具有非主键函数依赖的关系模式设计的参数化框架，从根本上将模式优化与操作层面的性能提升联系起来。

在第 3 节中，我们总结了组织模式的设计。第 4 节介绍了之前的工作算法，第 2 节介绍了参数化 3NF 规范化，而第 5 节展示了实验结果并进行了分析。我们在第 6 节中进行了总结，并对未来的相关工作进行了评论。相关材料可在 https://github.com/zzxhelloworld/3NF 获取，其中包括附录中提供了所有证明的扩展版本。

2 基础理论与前期工作

我们总结了构建框架所需的基础知识，首先从关系数据库设计中的概念入手[20, 26, 31]。示例 1.1 用于说明其中的大部分概念。

函数依赖、范式和规范化。关系模式是一个有限的属性集 R，每个属性 A 都有一个可能值的域 dom(A)。在 R 上的关系是来自笛卡尔积 $\hat{I}A \in R$ dom(A) 的有限元组子集。对于元组 t 和 R 的子集 $S \subseteq R$，t[S] 表示 t 在 S 上的投影。直观地说，关系形式化了基于列名的记录表。

函数依赖（FD）是一种表达式 $X \to Y$，其中属性集 X、$Y \subseteq R$。关系满足 $X \to Y$ 当且仅当每对在 X 的所有属性上值匹配的记录，在 Y 的所有属性上也值匹配。函数依赖 $X \to Y$ 是平凡的，当且仅当 $Y \subseteq X$。

对于函数依赖集 $D \cup \{d\}$，如果满足 D 中所有函数依赖的关系也满足 d，则称 D 推导出 d。D 的语义闭包是包含所有由 D 推导出的函数依赖的集合 D*，它等于通过诸如阿姆斯特朗公理 [3] 等公理化方法从 D 推导出的函数依赖的语法闭包 D+。如果 D+ = T+，则称函数依赖集 D 和 T 相互覆盖。

对于关系 R 上的函数依赖集 D，若不存在 X 的真子集 Z 使得 $Z \to Y \in D+$ 成立，则函数依赖 $X \to Y \in D+$ 是最小的。X 是关系 R 的键，当且仅当函数依赖 $X \to R$ 能由 D 推导得出。此时，$X \to R$ 是键依赖。若 $X \to R$ 不能由 D 推导得出，则函数依赖 $X \to Y$ 是非键依赖。关系 R 的键 X 是最小的，当且仅当 $X \to R$ 是最小的键依赖，即不存在 X 的真子集 Y 也是 R 的键。K 表示由 D 推导得出的最小键的集合。属性 $A \in R$ 对于 D 是主属性，当且仅当 A 包含在 R 的某个最小键中。

函数依赖（FD）编码了底层领域的业务规则，但也可能导致数据值出现冗余。实际上，对于关系 r 中满足函数依赖集 D 的元组 t，只要将 t[A] 的值 v 更改为任何不同的值 v' 都会导致关系违反 D 中的某些函数依赖，则数据值出现 v = t[A] 即为冗余。对于非平凡的函数依赖，当且仅当它不是键依赖时，才存在具有冗余数据值出现的关系[37]。

对于关系 R 上的函数依赖集 D，（R， D）处于 Boyce-Codd 范式（BCNF）当且仅当对于 D+ 中的每个非平凡函数依赖 $X \to A$，X 是 R 的键。因此，BCNF 表征了由函数依赖所导致的冗余数据值的不存在。数据冗余会导致更新效率低下，因为对于冗余出现的数据值的更新需要应用于每一个冗余出现的位置。如果这些值未被一致更新，就会出现完整性故障，即违反函数依赖。

R 的分解 D 是 R 的子集 S 的集合，满足 ÐS ∈D S = R。D 是无损的，当且仅当对于满足 D 的 R 上的每个关系 r，有 r = ▷◁S ∈D r[S]，即 r 是其投影 r[S] = {t[S] | t ∈r} 的无损连接。D 是依赖关系保持的，当且仅当 D+ = (S ∈D D[S])+ 其中

Ð

D[S] = {X →Y | X ∪Y ⊆S ∧X →Y ∈D+}。如果对于 D 中的每个 S，（S, D[S]） 都处于 BCNF（3NF）状态，则 D 处于 BCNF（3NF）状态。如果 D 不是依赖关系保持的，则某些函数依赖的完整性只能通过一些关系的昂贵连接来验证。虽然总是存在无损分解到 BCNF，但对于某些 （R， D） 并不存在无损且依赖关系保持的分解到 BCNF。

因此，关系模式 （R， D） 属于第三范式（3NF）当且仅当对于 D+ 中的每个非平凡函数依赖 X → A，X 是 R 的键或者 A 是主属性。实际上，总是可以将关系模式无损且保持依赖地分解为 3NF。然而，除非是 BCNF 分解，否则需要容忍数据冗余。经典算法 [28] 对于输入的 （R， D） 计算出无损且保持依赖的 3NF 分解，只要有可能就使其处于 BCNF。3NF 被证明在所有无损且保持依赖的分解中能保证最少的数据冗余源 [17]。然而，它们并未旨在打破 3NF 模式的平局。

复杂性结果。对于规范化中的计算问题，强调其下界复杂性是很重要的。例如，判定给定模式 （R， D） 是否满足第三范式 （3NF） 是 NP 完全问题 [14]，判定给定属性 A ∈ R 对于 （R， D） 是否为候选键也是 NP 完全问题 [4, 24]，判定是否存在无损且保持依赖的 BCNF 分解是 NP 难问题 [35]。计算模式 （R， D） 的最小键数量是 #P 完全问题 [12]，但存在一种算法可以在输出大小的线性时间内计算出最小键的集合 [24]。虽然最小键的数量可能随给定函数依赖的数量呈指数增长，但在实际应用中这种情况很少发生，因此通常可以高效地计算出最小键的集合 [24]。

对 BCNF 进行参数化。近期的研究通过最小键的数量 k 对 BCNF 进行了参数化[42]。在此，k 同时衡量更新和查询的复杂度。k 值越大，在更新过程中需要维护的唯一索引就越多，但更多的查询可以从这些索引中获益。从根本上说，由 k 个最小键组成的集合 K 构成了一个 k 级复合对象，其特征在于 k 唯一性和 k 依赖性[42]。其中，k 唯一性意味着 k 个最小键中的每一个都能唯一标识每个关系中的记录，而 k 依赖性则意味着维护所有 k 个最小键就足以在更新过程中保持数据完整性。例如，在示例 1.1 中，对于模式 （R1, D1），{EST， ESV， STV} 构成了一个 k = 3 级的复合对象。相反，在示例 1.1 中，对于模式 （R3, D3），{ESV， EMS} 不是一个 k = 2 级的复合对象：虽然它满足 2 唯一性，但不满足 2 依赖性，因为维护这两个键 ESV 和 EMS 并不足以维持非键依赖关系 VS → M。模式 （R， D） 处于 k 级复合对象范式 （k-CONF） 当且仅当 D+ 中每个最小函数依赖的左部都属于一个复合对象。在 BCNF 中，所有约束均由最小键强制执行，而在 k-CONF 中，所有约束均由 k 个最小键强制执行。因此，k 可以在 BCNF 模式之间打破平局[42]。

从计算角度来看，最佳的经典算法[28]通过首先消除具有较多（较少，分别对应）最小键的冗余 BCNF 模式进行了优化，目的是在分解中处于 BCNF 的那些模式上最小化更新复杂度（最大化查询效率，分别对应）。

然而，对于完整性维护造成最大瓶颈的非主键函数依赖尚未得到考虑，这正是我们当前工作的贡献所在。实际上，我们将使用 k 和适当覆盖中的非主键函数依赖的数量 f 对 3NF 进行参数化，从而得到 (k, f)-3NF，其中 f = 0 的特殊情况涵盖了 k-CONF。

3 参数化 3NF 的基础

我们将通过获取能够优化规范化的参数来改进 3NF 框架。作为副产品，3NF 和 BCNF 之间的差异也将变得可量化。我们将改进

维护经典 3NF 的定义取决于在更新过程中保持最小键和非键 FD 所需的努力。将非键 FD 隔离出来使我们能够在规范化过程中将其开销降至最低。这种最小化从逻辑层面延伸到操作层面，从而降低完整性维护的开销。

我们直观地概述这一部分。第三范式（3NF）自然与最小键集 K 相关联，因为每个非平凡函数依赖 X → A 的左部 X 要么包含某个最小键，要么右部 A 是某个最小键的元素（即 A 是主属性）。因此，K 至少有两个用途：1）每个最小键都会产生一个唯一的索引；2）K 使我们能够分离出那些不能由键强制执行的函数依赖。将函数依赖集 D 分解为 K 和 F 的适当覆盖，其中 F 是非键函数依赖集的覆盖，这将正式引导我们定义 D 的 3NF 核心。如果该核心是 D 的覆盖，则 D 在 3NF 中。基于 K 的基数 k 和 F 的最小约简覆盖的基数 f，我们可以比较处于 (k, f)-3NF 中的模式。如果 f = 0，则模式处于 BCNF 中，更确切地说处于 k-CONF 中。因此，f 衡量了处于 (k, f)-3NF 中的模式相对于处于 k-CONF 中的模式的开销。

3.1 非及物复合对象

在 BCNF 的特殊情况下，集合 K 构成了一个 k 级的复合对象。因此，BCNF 模式在更新期间仅需要 k 个最小键来维护完整性。在一般情况下的 3NF 中，还需要非键的函数依赖，但仅针对右部属性为候选键属性的情况。因此，我们将复合对象推广为非传递复合对象，将其定义为在更新期间足以维护其输入函数依赖集完整性的 3NF 子结构。

形式上，设 （R， D） 表示关系模式 R 及其上的一组函数依赖 D。设 T 表示 R 上的一组函数依赖，满足

$$\mathcal{T} \subseteq \{X \subseteq R \mid X \to R \in \mathcal{D}^+\} \cup \{X \to Y \in \mathcal{D}^+ \mid (X \to R \notin \mathcal{D}^+) \wedge (Y - X \subseteq \mathcal{P})\}$$

哪里

$$\mathcal{P} = \{A \in R \mid \exists K \to R \in \mathcal{D}^+ \wedge \forall K' \subset K (K' \to R \notin \mathcal{D}^+) \wedge A \in K\}$$

表示 D 的主属性集。我们称 T 为（R，D）的 3NF 子结构。实际上，T 由键和非键的函数依赖组成，其右侧属性均为主属性。因此，3NF 子结构满足 3NF 的要求。然而，它们可能无法强制执行输入集中的所有函数依赖。这一额外特性是特殊的，并定义如下。

定义 3.1（非传递复合对象）。设 （R， D） 表示关系模式 R 及其上的一组函数依赖 D。设 T 表示 （R， D） 的 3NF 子结构。当且仅当满足以下条件时，我们称 T 为 D 的非传递复合对象：

· （第三范式更新完整性）

对于所有满足 D 的 R 上的关系 r，对于所有 t ∈ dom(R)，如果 r ∪ {t} 满足 T，则 r ∪ {t} 满足 D。

□

因此，当 D 的所有非传递组合对象中的函数依赖都有效时，D 的完整性得以保留。接下来，我们对这些定义进行说明。

例 3.2. 考虑 R = {E， M， S， T} 和 D = {ET →MS, M →E}。最小键集为 K = {ET, MT}，属性集 P = {E， M, T}，唯一非主属性为 S。因此，（R， D） 处于 3NF。然而，T ‘ = {ET， MT， MS →E} 不是 D 的传递复合对象（因为 D 中的函数依赖 M →E 不由 T’ 推导得出）。不过，T = T' ∪{M →E} 是 D 的传递复合对象。实际上，满足 ET 的每个关系也必然满足 ET →MS。然而，{ET， MT} 不是复合对象。这一点可以从以下记录中观察到。

	事件	时间	经理	状态
t':	车间	21/11/2024	索菲	批准的
t:	研讨会	19/12/2025	索菲	拒绝；下降；衰退

确实，r = {t '} 满足 D，而 r ∪{t} 满足 ET 和 MT，但不满足 M → E。因此，{ET， MT} 不是一个复合对象。□

3.2 非及物复合对象范式

非传递组合对象并非唯一。实际上，3NF 子结构可能包含非最小键或非最小函数依赖。类似于 BCNF 中由键 K 给出的唯一组合对象，我们现在通过将输入函数依赖集 D 分割为 K 和所有非键函数依赖的集合 F 来定义唯一的 3NF 子结构，其中 F 中的函数依赖 X → A ∈ D+ 的右部 A 是主属性，且 X 是最小的，不存在 X 的真子集 Y 使得 Y → A ∈ D+ 成立。

定义 3.3.（3NF 核）对于关系模式 R 上的函数依赖集 D，我们用 K = {X ⊆ R | X → R ∈ D+ ∧ ∀Z ⊂ X(Z → R < D+)} 表示由 D 所蕴含的最小键的集合，并且

$$\mathcal{F} = \{Z \rightarrow A \in \mathcal{D}^+ \mid (Z \rightarrow R \notin \mathcal{D}^+) \wedge (A \in \mathcal{P} - Z) \wedge (\forall Y \subset Z(Y \rightarrow A \notin \mathcal{D}^+))\}$$

表示由 D 所蕴含的右部为非主属性的非主键最小函数依赖集。我们称 K ∪ F 为 D 的 3NF 核。

□

以下内容承接我们之前的示例，移除了一个非最小非主键函数依赖。

Example 3.4. For $R = \{E, M, S, T\}$ and $\mathcal{D} = \{ET \rightarrow MS, M \rightarrow E\}$ from Example 3.2, $\mathcal{T}_c = \mathcal{K} \cup \mathcal{F}$ forms the 3NF-core of $\mathcal{D}$ where $\mathcal{K} = \{ET, MT\}$ and $\mathcal{F} = \{M \rightarrow E\}$. □

3NF 核 f 是非主键唯一函数依赖的集合，通过选择来表示某些最小化简的 F 。这将在下一节用于最小化以优化 3NF 合成。不过，目前我们将 CONF [42] 对 BCNF 的最新特征描述推广到通过 3NF 核对 3NF 的特征描述。我们已经完成了所有工作，因为 3NF 核只需满足 3NF 更新完整性即可捕获 3NF 。

定义 3.5.（不可传递复合对象范式）
设 D 表示关系模式 R 上的函数依赖集。那么，（R， D） 处于非传递复合对象范式（iCONF）当且仅当 D 的 3NF 核是 D 的非传递复合对象。□

定义 3.5 捕获了这样一个特殊情况：当关系模式 （R， D） 处于 k 级复合对象范式时，其 3NF 核 K ∪ F 不仅是无传递的复合对象，而且是 k 级复合对象，即不包含任何非平凡函数依赖且 K 由 k 个最小键组成。实际上，在这种情况下，|K| = k 意味着 K 满足 k 唯一性，而 F = ∅ 则意味着 3NF 更新完备性蕴含 k 依赖性。

定义 3.5 与 D 的表示方式无关。实际上，对于 D 的每一个覆盖 T，（R， D） 属于 iCONF 当且仅当 （R， T） 属于 iCONF。现在我们将说明 iCONF 的定义。

例 3.6. 对于例 3.4 中的 R = {E， M， S， T} 和 D = {ET → MS, M → E}，其 3NF 核 Tc 满足 3NF 更新完备性，因此 （R， D） 属于 iCONF。

□

3.3 第三范式和非传递复合对象范式

我们现在将建立 3NF 与 iCONF 之间的等价关系。其主要思想在于认识到 3NF 子结构 T 是 3NF 更新完备的，当且仅当输入的函数依赖集处于 3NF 形式且被 T 覆盖。

引理 3.7.（主要引理）设 D 表示关系模式 R 上的 FD 集合。设 T 表示（R，D）的一个 3NF 子结构。则 T 是 3NF 更新完备的，当且仅当以下两个条件成立：（1）D ⊆T + ；（2）（R，D）处于 3NF 状态。

□

我们现在得出结论：处于第三范式的模式的第三范式核必然是一个非传递的复合对象。

推论 3.8. 设 D 表示关系模式 R 上的函数依赖集。若 （R， D） 处于第三范式，则 3NF 核 K ∪ F 是 （R， D） 的非传递复合对象。

□

正如预期的那样，我们可以用 iCONF 来描述 3NF。

定理 3.9. 对于所有关系模式 R 及其上的一组函数依赖 D，（R， D） 属于第三范式当且仅当 （R， D） 属于 iCONF。

□

定理 3.9 通过 3NF 核建立了 3NF 的结构特征，我们将利用最小键和非键函数依赖的参数来探索其对优化 3NF 合成的应用。因此，我们为参数化 3NF 规范化奠定了基础。文献[42]中的最新成果现在成为我们新框架的特例。特别是，k-CONF 是 iCONF 的特例，其中给定的函数依赖集 D 被 k 个最小键的集合 K 所覆盖。

推论 3.10. 设 D 表示关系模式 R 上的一组函数依赖。则以下陈述等价：

关系模式 D 的第三范式核被由 k 个最小键组成的集合 K 所覆盖。
（2）（R，D）处于 BCNF 形式，且具有 k 个最小键。
（3）（R，D）处于 k 级别置信度中。

□

接下来我们说明如何在不同的第三范式模式之间打破平局，这是之前的工作所无法做到的。

例 3.11. 考虑以下两个模式，它们属于例 1.1 中 （R， D） 分解的模式：

- R3 = EMSV 和 D3，具有非主键函数依赖 VS → M 以及两个最小键 ESV 和 EMS
- R4 = EMST 和 D4，具有两个非主键的函数依赖 MT → E、ET → M 以及三个最小键 EST EMS 和 MST。

要么（R3，D3）要么（R4，D4）是多余的，所以我们可以在分解时选择包含哪一个。D3 包含一个最小非主键函数依赖和两个最小主键，而 D4 包含两个最小非主键函数依赖和三个最小主键。如果我们倾向于拥有更少的非主键函数依赖，那么我们选择（R3，D3）而非（R4，D4），但如果我们倾向于拥有更多的最小主键，那么我们选择（R4，D4）而非（R3，D3）。

规范化。最后一个示例说明了如何通过使用参数（例如非主键函数依赖的数量或最小键的数量，甚至它们的组合）来优化经典规范化，从而进一步打破平局。因此，所得到的算法针对特定策略，而不是在冗余模式之间随意选择。下一节将建立我们的参数化框架

4 参数化 3NF 规范化

在所有无损且保持依赖关系的分解中，满足第三范式（3NF）的分解产生的数据冗余最少，但并非所有处于 3NF 的模式都是相同的，非主键函数依赖会导致严重的更新开销。这为规范化提供了巨大的机会，令人惊讶的是，此前尚未有相关研究涉及。最近，根据所具有的最小键的数量对 BCNF 模式进行了分类，但尽管存在非主键函数依赖，3NF 模式却仍未受到关注。

我们已经将给定函数依赖集 D 中尽可能多的应用语义转移到了最小键集 K 中，从而得到了最小的非键函数依赖集 F。我们将使用最小化简化的

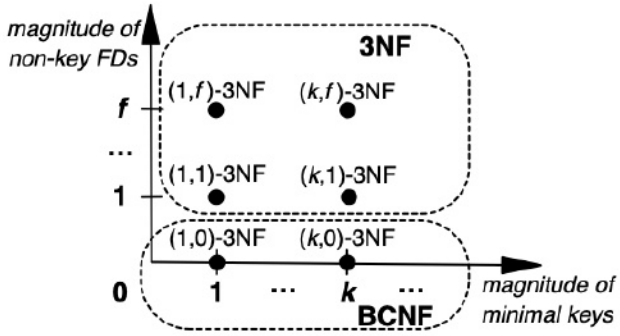

图 3. 通过参数区分不同的第三范式模式

（最优）覆盖以使其基数（大小，分别而言）最小化 F。这使我们能够引入 3NF 的参数化变体，研究参数化规范化的计算复杂性，引入一套算法并用我们的示例进行说明。

4.1 参数化第三范式

计算的第一步是求解非主键函数依赖集 F 的二次最小化问题[25]。迈尔[25]提出了最小化基数和大小的最小化简集和最优覆盖的概念。虽然最优覆盖能提供给定函数依赖集在任何可能表示中出现的属性总数最少，但其基础决策问题是 NP 完全问题[25]。相比之下，最小化简集能提供在任何表示中出现的函数依赖最少且无冗余属性出现的覆盖。

对于处于第三范式（3NF）的关系模式 （R， D），其最小键集 K 以及由 D 所蕴含的非主键函数依赖集的最小约简覆盖 Fc，我们称 （K, Fc） 为 （R， D） 的 3NF 核的最小约简覆盖。对于由 D 所蕴含的非主键函数依赖集的最优覆盖 Fs，我们称 （K, Fs） 为 （R， D） 的 3NF 核的最优覆盖。

定义 4.1. 设 D 是 R 上的函数依赖集，kc 、ks 为正整数，fc 、fs 为非负整数。若关系模式 （R， D） 属于 3NF 且其 3NF 核的某个最小约简覆盖 （K, Fc ） 中 K 的基数为 kc ，Fc 的基数为 fc ，则称 （R， D） 属于 (kc, fc)-3NF 。若关系模式 （R， D） 属于 3NF 且其 3NF 核的某个最优覆盖 （K, Fs ） 中 K 的大小为 ks ，Fs 的大小为 fs ，则称 （R， D） 属于 (ks, fs)-3NF 。 □

例 4.2. 图 1 以基数为基础的参数来说明定义 4.1：（R1, D1） 处于 (3, 0)-3NF，（R2, D2） 处于 (2, 0)-3NF，（R3, D3） 处于 (2, 1)-3NF，（R4, D4） 处于 (3, 2)-3NF，（R5, D5） 处于 (1, 0)-3NF。由于所有约束集都是其自身的最优覆盖，因此可以根据模式的大小对其进行分类：（R1, D1） 处于 (9, 0)-3NF，（R2, D2） 处于 (4, 0)-3NF，（R3, D3） 处于 (6, 3)-3NF，（R4, D4） 处于 (9, 6)-3NF，（R5, D5） 处于 (2, 0)-3NF。 □

当写成 （k, f） 时，我们指的是 (k, f) ∈{(kc, fc), (ks, fs)}，因此我们同时处理最小约简覆盖和最优覆盖这两种情况。关系模式 （R， D） 处于 (k, f)-3NF 的性质与语义约束的表示方式无关。也就是说，如果 D ‘和 D 相互覆盖，那么 （R， D） 处于 (k, f)-3NF 当且仅当 （R， D’ ） 处于 (k, f)-3NF。所有最小约简覆盖的基数相同，所有最优覆盖的大小也相同。因此，分别找到某个最小约简覆盖或最优覆盖，就能保证 f 是可达到的最佳值。

图 3 展示了参数 k 单独如何区分 BCNF 模式，组合 （k, f） 如何区分 3NF 模式，以及具有最小键 k 的 BCNF 模式如何是 3NF 的特殊情况。

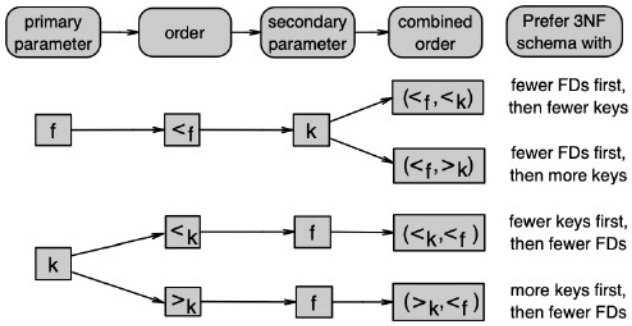

图 4. 参数和阶数的选择

具有 k 个最小键和 f = 0 个非键外键依赖的模式。约束集的规模指的是其基数或大小。

4.2 打破 3NF 模式之间的联系

我们现在利用这些参数来阐述打破 3NF 模式之间平局的策略，之后针对特定目标优化 3NF 合成。通过启用多种策略，我们使用参数 k ∈{kc，ks } 和 f ∈{fc，fs } 来定义不同的有限顺序 O，例如：i) 最小化或最大化 ii) 最小键集或非键 FD 集的 iii) 卡片数或大小。我们总是倾向于选择较小而非较大的非键 FD 集，因此始终要最小化 f 。

基于给定的排序 O，我们确定一个排序 <O，它按照 O 的偏好顺序对元素进行排序。在 <O 中，最小的元素总是被视为“最佳”，因为我们的规范化框架旨在最小化完整性维护。例如，如果 O 是 <f，那么 <O 就是非负整数 0，. . .，n 的自然顺序 0 < . . . < n；如果 O 是 >k，那么 <O 就是 0，. . .，n 的逆自然$_{\text{顺序}}$ n < n − 1 < . . . < 1 < 0。

示例 4.3. 在示例 3.11 中的 （R3, D3） 和 （R4, D4） 上，订单 O1 倾向于更少的非主键函数依赖（<fc），因此 R3 中的函数依赖 1 排在第 1 位，R4 中的函数依赖 2 排在第 2 位，得出 1 < 2。订单 O2 倾向于更多的最小键（>kc），因此 R4 中的键 3 排在第 1 位，R4 中的键 2 排在第 2 位，得出 3 < 2。

□

图 4 展示了选择主参数和次参数如何得出组合订单 O，分别为（<f，<k）、（<f，>k）、（<k，<f）和（>k，<f）。接下来的例子将明确此类订单 O 的排序 <O。

示例 4.4. 我们从具有主参数和次参数的不同排序 O 中获得的排名如图 1 所示。对于第一个排名，具有空函数依赖集且最小键集大小为 k0 的模式排在首位。主排序 <f 表明具有较少函数依赖的模式优先级更高，而剩余的并列情况则由次排序 >k 来解决，该排序优先考虑具有更多键的模式。对于第二个排名，具有空函数依赖集且最小键集大小为 1 的模式排在首位。□

例 4.5. 在例 3.11 中的 （R3, D3） 和 （R4, D4） 上，排序 O3 = (<fc, >kc) 首先倾向于选择非主键函数依赖较少的元素，然后倾向于选择主键较少的元素，因此 R3 中的元素 （1, 2）（具有 1 个函数依赖和 2 个主键）排在第 1 位，而 R4 中的元素 （2, 3）（具有 2 个函数依赖和 3 个主键）排在第 2 位，从而得到排序 (1, 2) < (2, 3)。排序 O4 = (>kc, <fc) 首先倾向于选择主键较多的元素，然后倾向于选择非主键函数依赖较少的元素，因此 R4 中的元素 （2, 3） 排在第 1 位，而 R3 中的元素 （1, 2） 排在第 2 位，从而得到 (2, 3) < (1, 2)。□

表 1. 来自示例 4.4 的排名

O	零
(<f, >k): $(0,k_0)$	... < $(_{0,}$ 1)... < (f ,k_f)... < (f, 1)
(f, k)：	（$0,$ 1）⋯⋯< $(0,k_0)$ . . . < (f, $_{1)}$ . . < (f ,k_f)
(< k , < f) : (1 , $_{0}$) . . . < (1 , f_1) . . . < (k , $_{0}$) . . . < (k , f k)	
(>k, <f): $(k, _{0)}$	⋯ < (k, fk) ⋯ < $(1, _{0)}$ ⋯ < (1, f_1))

关键模式（f > 0）可能对参数 k 的应用顺序与 BCNF 模式（f = 0）不同。例如，我们可以在 BCNF 模式上使用 <k 来最小化 k，而在关键模式上使用 >k 将 k 作为次要参数来最大化。我们用 <O’-BCNF 表示 f = 0 时来自 O’-BCNF 序的元素排序，用 <O”-3NF 表示 f > 0 时来自 O”-3NF 序的元素排序。我们通过定义前者中最不受欢迎的元素排在后者中最受欢迎的元素之前，将 <O’-BCNF 和 <O”-3NF 合并为排序 <O-BCNF/3NF。如果 O’-BCNF 是 <k 而 O”-3NF 是 （<f, >k），则合并后的排序 <O-BCNF/3NF 如下。

$$\underbrace{1 < \ldots < k}_{<O'\text{-}BCNF} < \underbrace{(1,k_1) < \ldots < (1,2) < \ldots < (f,k_f) < \ldots < (f,2)}_{<O''\text{-}3NF}$$

请注意，关键模式至少有两个不同的最小键。我们现在将任何排序 <O 提升为 3NF 模式的排序 <OR，从而能够打破它们之间的平局。

定义 4.6. 对于处于第三范式（3NF）的模式 （R， D） 以及有限序 O 的排序 <O，我们定义 （R， D） 的 3NF 排序 rOR 为排序 <O 中使 （R， D） 处于 (k, f) - 3NF 的任何 （k, f） 的最小排序。我们进一步定义 3NF 模式的序 <OR 为 （R， D） <R （R′, D′） 当且仅当

O

哦，哦，哦，哦，哦。 □

接下来，我们继续以运行示例为例，针对不同的顺序比较不同的第三范式模式。

例 4.7. 我们继续例 4.3 和 4.5 的内容。对于订单 O ∈{O1，O3}，我们得到 3NF 秩 rOR3 = 1 和 rOR4 = 2，因此当更倾向于较少的函数依赖时，如预期的那样 (R3, D3) <OR （R4, D4）。然而，对于

O ∈{O_2,O_4} 我们有 3NF 等级 r_{OR_3}= 2 和 r_{OR_4}= 1，所以 (R_4， D_4) <O 如预期的那样，R′(R3, D3

′

我们倾向于更多的键。这两种模式都是关键的，因此将这些排序与任何< O-BCNF 合并都不会改变结果。

□

4.3 计算复杂度

我们分析的问题是确定给定模式是否属于（kc，fc 决策）-3NF 或（ks，fs 关联）-3NF，这与 3NF 数据库设计相关。这将有助于我们理解所开发算法的局限性，并为之后将要展示的实验结果设定预期。首先，我们来探讨基本问题。

参数化三范式
输入：（R， D），非负整数 (k, f) ∈ {(kc, fc), (ks, fs)} 问题：判定 （R， D） 是否处于 (k, f)-3NF 形式

由于计算最小键集的问题是输出多项式的，并且在实践中通常效率很高，因此我们可以将此集视为附加输入，特别是在我们的框架中，我们将此集与非键 FD 集分开。

带最小键集的参数化三范式
（R，D），非负整数 f ∈ {fc，fs} 设正整数集合 K 中的 k 属于 {kc， ks} 问题：判断（R，D）是否属于（k，f）-3NF

上述两个问题验证了一个模式是否满足参数化第三范式条件。虽然最近已证明，判定给定模式（R，D）是否处于 k-CONF（恰好具有 k 个最小键的 BCNF）的问题在输入规模上是多项式的[42，推论 5.2]，但对于基于基数和大小的变体，参数化 3NF 是 NP 完全问题。因此，除非 P = NP，否则测试范式条件本身是不可行的。因此，优化 3NF 模式设计受此下界限制。

NP 完全性源于判定给定属性是否为候选键的 NP 完全性[24， 定理 2]，而后者又基于这样一个事实，即对于给定的一组函数依赖，可能存在指数数量的键[24]。这也是判定对于模式（R，D）的一个真子集 S ⊂ R，（S，D[S]）是否处于 BCNF 为 coNP 难问题的原因[4， 推论 3]（具有 k 个最小键[42， 定理 5.3]）。

优化 3NF 模式设计的目标是无损且依赖关系保持的分解为 BCNF。实际上，如果无损分解不保留某些函数依赖或包含冗余的关键模式，则可以改进数据完整性维护。然而，即使不给定任何参数，也很难判定是否存在最优分解，这一点在文献[35， 定理 4]中已有说明。

参数化最优三范式设计
（R，D），非负整数 k ∈ {kc，ks} 问题：判断是否存在无损且保持依赖关系的分解为 (k, 0)-3NF 。

接下来，我们总结上述参数化决策问题的计算复杂度的结果。

定理 4.8. 与第三范式相关的参数化变体具有以下计算复杂度。

基于基数和基于大小的参数化 3NF 问题均为 NP 完全问题。

（2）基于基数的参数化 3NF 与最小键集问题是多项式时间可解的，但基于大小的参数化 3NF 与最小键集问题是 NP 完全问题。

（3）基于基数和基于大小的每种变体的参数化最优 3NF 设计均为 NP 难问题。

□

由于问题中的变量是在模式级别定义的，因此其实例通常规模较小。因此，尽管存在最坏情况下的指数级界限，但在实际计算中通常效率很高。如果不要求依赖关系保持，则可以在确定性时间内获得无损 BCNF 分解，其时间复杂度为输入规模的多项式 [35]。

4.4 算法套件

我们现在将把我们的想法转化为一种算法，该算法采用特定策略来优化最先进的归一化方法。

维护先前非主键保证函数依赖的算法。一种无损且依赖关系保持的分解为第三范式的方法，只要有可能就分解为 Boyce-Codd 范式 [28]。只有当每个关键模式都是冗余的时，才返回 BCNF。最近 [42]，通过将 BCNF 模式按其展现的最小键的数量进行参数化，改进了这一点。这使得在 BCNF 模式之间打破平局成为可能。然而，还没有工作尝试在关键模式之间打破平局。因此，输出未针对完整性进行优化。

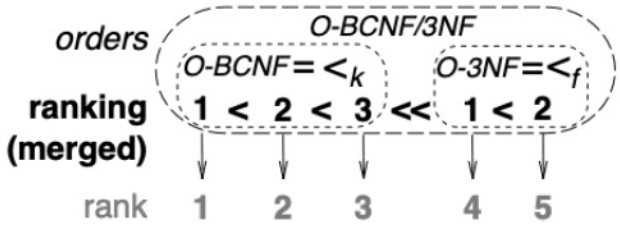

图 5. 例 4.9 中 <O-BCNF/3NF 的排序

我们将通过引入无损且依赖关系保持的 3NF 分解的共词序 <O-BCNF/3NFD 来解决这一优化机会，使用 O-BCNF 和 O-3NF 的排序 <O-BCNF/3NF 作为参数 k 和 f 。我们的算法将返回一个无损且依赖关系保持的 3NF 分解，该分解对于目标词序 <O-BCNF/3NF 是最优的。简单来说，我们首先尽量减少根据 <O-3NF 排序较低的关键模式的数量，然后尽量减少根据 <O-BCNF 排序较低的 BCNF 模式数量。

例 4.9. 考虑 D = {D1, D2}，其中 D1 和 D2 来自例 1.1。O-BCNF 的 <kc 关系和 O-3NF 的 <fc 关系得出以下合并排序 <O-BCNF/3NF，其中 O-BCNF 和 O-3NF 由 << 分隔：1 < 2 < 3 << 1 < 2，如图 5 所示。□

现在我们将介绍一种按模式等级对模式进行分类的共词序。直观地说，当一个分解在它们不同的最差等级上不包含任何模式时，它就优于另一个分解。在示例 4.9 中，D1 和 D2 不同的最差等级是 5，由于 D1 在等级 5 上没有任何模式（即其最小约简覆盖中有 2 个非主键函数依赖的那些模式），所以它优于 D2。其形式定义如下。

定义 4.10. 设 D 为模式 （R， D） 分解为 3NF 的无损且保持依赖的分解集合。对于 D 中的 D ∈D，对于跨越 D 中任何分解中出现的所有参数值的 <O-BCNF/3NF 排序，设排序 r 在 <O-BCNF/3NF 中，令 SrD 表示集合 D 中具有 rOS = r 的模式 （S, D[S]） 的集合，即

$$\mathcal{S}_r^{\mathbb{D}} = \{(S, \mathcal{D}[S]) \in \mathbb{D} \mid (S, \mathcal{D}[S]) \text{ has rank } r_O^S = r \text{ in } <_{O\text{-}BCNF/3NF}\}$$

并且设 nrD 为其基数，即 nrD = |SrD|。对于 D′， D′′∈D，D′ 在 D 上优于 D′′（D′<O-BCNF/3NFD D′′）当且仅当对于最差的秩 r 在 <O-BCNF/3NF 中，′′SrD′， SrD′′， SrD = ∅。 □

或者，D′在 n-better 方面不如 D′′（D′<nO-BCNF/3NF D′′）当且仅当在 <O-BCNF/3NF 中最差的秩 r 有 nrD′=0 或 nrD′<0，且 nrD′=0。然而，如果 D′在 D-better 方面优于 D′′，那么 D′也 n 优于 D′′，但反之则不然。由于我们的算法将确保 <O-BCNF/3NFD 优化性，因此它对于 <nO-BCNF/3NF 也是最优的。

为了说明我们的定义，我们将运行示例在不同顺序下的分解进行比较。

例 4.11. 考虑 D = {D1, D2}，其中 D1 和 D2 来自例 1.1。首先，令 O-BCNF 为 <kc ，O-3NF 为 <fc ，如例 4.9 所示，见图 5。下表列出了 D 中每个 D ∈D 以及每个秩 r 对应的 SrD 。

D	S1D	S2D	三维立体S3D	S4D	S5D
D_1	∅	$\{R_2\}$	$\{R_1\}$	$\{R_3\}$	∅
D_2	$\{R_5\}$	∅	$\{R_1\}$	∅	$\{R_4\}$

Algorithm 1 ıCONF(R, $\mathcal{D}$, *O-3NF*, *O-BCNF*)

Require: $(R, \mathcal{D})$ with FD set $\mathcal{D}$ over schema R, 3NF order *O-3NF*, BCNF order *O-BCNF*
Ensure: Lossless, FD-preserving 3NF decomposition $\mathbb{D}$ of $(R, \mathcal{D})$ that is $<^{D}_{\text{O-BCNF/3NF}}$-optimal

```
Compute the atomic cover \overline{\mathcal{D}}_a of \mathcal{D} [15, 28]
\mathcal{D}_a ← \overline{\mathcal{D}}_a, \mathcal{D}_c ← ∅
for all X → A ∈ \mathcal{D}_a do
    for all Y → B ∈ \mathcal{D}_a (YB ⊆ XA ∧ XA ⊈ Y^+_{\mathcal{D}}) do
        Compute k_XA and f_XA in target 3NF-core of \mathcal{D}_a[XA]
        \mathcal{D}_c ← \mathcal{D}_c ∪ {(X → A, k_XA, f_XA)}
\mathbb{D} ← ∅;
for all (X → A, k_XA, f_XA) ∈ \mathcal{D}_c in reverse <_O-3NF-ranks do
    if \overline{\mathcal{D}}_a\{X → A} ⊭ X → A then
        \mathbb{D} ← \mathbb{D} ∪ {(XA, \mathcal{D}_a[XA], k_XA, f_XA)}
    else
        \overline{\mathcal{D}}_a ← \overline{\mathcal{D}}_a\{X → A}
for all X → A ∈ \overline{\mathcal{D}}_a\\mathcal{D}_c do
    k_XA ← |{K | K → XA ∈ \mathcal{D}_a[XA])^+}| using [24]
    \mathcal{D}_a ← (\mathcal{D}_a\{X → A}) ∪ {(X → A, k_XA)}
for all (X → A, k_XA) ∈ \overline{\mathcal{D}}_a\\mathcal{D}_c in reverse <_O-BCNF-ranks do
    if \overline{\mathcal{D}}_a\{X → A} ⊭ X → A then
        \mathbb{D} ← \mathbb{D} ∪ {(XA, \mathcal{D}_a[XA], k_XA)}
    else
        \overline{\mathcal{D}}_a ← \overline{\mathcal{D}}_a\{X → A}
Remove all (S, \mathcal{D}_a[S]) ∈ \mathbb{D} if ∃(S', \mathcal{D}_a[S']) ∈ \mathbb{D}(S ⊆ S')
if there is no (R', \mathcal{D}') ∈ \mathbb{D} where R' → R ∈ \mathcal{D}^+ then
    Choose a minimal key K for R with respect to \mathcal{D}
    \mathbb{D} ← \mathbb{D} ∪ {(K, \mathcal{D}_a[K], 1)}
Return(\mathbb{D})
```

D1 和 D2 的模式不同的最差等级是第 5 级。因为 nD1 = 0 < 1 = nD2 ^{5}，我们首先，令 D1 <DO-BCNF/3NF D2。其次，令 O’-BCNF 为 <kc 且 O’-3NF 为 >kc，从而得到以下排序 <O’-BCNF/3NF，其中 O-BCNF 和 O-3NF 由 << 分隔：1 < 2 < 3 << 3 < 2。下表显示了对于 D ∈D 中的每个 D 以及每个等级 r 的 SrD。

D	S1D	S2D	三维立体S3D	S4D	S5D
D_1	∅	$\{R_2\}$	$\{R_1\}$	∅	$\{R_3\}$
D_2	$\{R_5\}$	∅	$\{R_1\}$	$\{R_4\}$	∅

D1 和 D2 在模式上出现差异的最差等级是第 5 级。因为 nD1 = 1 > 0 = nD2 ^{5}，我们令 D2 <OD’-BCNF/3NF D1。□

算法 1 计算出一种无损且保持依赖关系的 3NF 分解，该分解属于 <O-BCNF/3NFD - 对于其输入而言是最优的。它从输入 D 的原子覆盖 Da 开始（第 1 行），即由 D 所蕴含的唯一一组最小函数依赖 X → A [15, 28]。第 2 行创建原子覆盖的一个副本，之后可从中移除冗余的函数依赖，而原始的闭包则用于检查关键函数依赖（第 4 行）。

第（3 - 6）行的部分称为关键部分。它汇集了关键模式集 Dc（第 6 行）以及确定输入顺序 O-3NF 的 <O-3NF 排序的其参数 k 和 f 的值（第 5 行）。

第 7 至 12 行的部分称为 Opt-3NF，在此会按 <O-3NF 的逆序对导致关键模式的函数依赖进行评估（第 8 行）。如果某个函数依赖是冗余的（第 12 行），则无需该模式。从最差到最佳处理这些函数依赖，可确保在可能的情况下消除剩余的最差模式。如果函数依赖不是冗余的（第 9 行），则添加该模式（第 10 行）。Opt-3NF 确保了 <O-3NF 的最优性。

第（13 - 15）行的部分称为“关键”。它计算非关键模式的最小键的数量。该计算可在输出多项式时间内完成[24]。

在 O-BCNF 中，第 16 行的部分（即第 16 至 20 行）称为 FD Opt-BCNF。当存在冗余循环（通过第 20 行）或非关键时，它会以相反的顺序添加其模式中的 FD（第 17/18 行）。这消除了遵循 O-BCNF 的冗余 BCNF 模式。

第 21 行中的部分称为子集。在此，我们移除任何是另一个模式子集的模式。

算法 最后，第 1 部分在第 22 至 24 行被称为无损。它在输出中添加了一个最小键，以确保在第 25 行返回时分解是无损的。至此，解释完毕。

以下结果改进了现有技术[42]，该技术返回无损且保持依赖关系的分解，分解为 3NF 形式，如果可能的话则为 BCNF 形式，并且在那种情况下为 <kc - 最优。

定理 4.12. 对于输入 （R， D, O-3NF, O-BCNF），算法 1 返回一个无损且依赖关系保持的分解为 3NF 的结果，该结果是 <O-BCNF/3NFD-最优的。

□

与以往任何工作不同，算法 1 优化了关键模式，并且针对任何给定的目标策略进行优化。算法 1 的运行时间在最坏情况下呈输入的指数级增长。这是由定理 4.8 中的 NP 难结果得出的。实际上，存在无损且保持依赖关系的分解为 BCNF 等价于存在一个原子覆盖，其中每个关键模式都是冗余的 [15, 28]。

4.5 运行示例

我们通过将 $_{FC}$ 作为 O3N 算法 $_{F}$ 和将 KC1 作为 O 展示 $_{BCNF}$ F 来说明，如何最小化非关键属性依赖分解的数量以及 BCNF 模式的最小键的数量。首先，我们来看示例 1.1 中关键派生模式的分解。

我们从原子覆盖集 EMS →V 、SV →M、EMS →T、MT →E、ET →M、MST →V 、SET →V 、ESV →T、STV →E 开始。

关键模式为（R3，D3），具有 1 个非主键函数依赖和 2 个最小键；（R4，D4），具有 2 个非主键函数依赖和 3 个最小键；以及（R6 = MSTV，D6），具有 1 个非主键函数依赖 SV → M 和 2 个最小键 STV 和 MST。BCNF 模式为（R5，D5），具有 1 个最小键；（R2，D2），具有 2 个最小键；以及（R1，D1），具有 3 个最小键。

对于 <fc 作为 O3N $_{F}$，关键模式的排序为：(R3, D3) = Rfc (R6, D6) < Rfc （R4, D4）。由于生成 R4 的函数依赖 EMS → T 是冗余的，因此模式 （R4, D4） 不需要。生成 R3 的函数依赖 EMS → V 现在不再冗余，所以模式 （R3, D3） 被添加到分解中。接下来，生成 R6 的函数依赖 MST → V 仍然是冗余的，因此模式 （R6, D6） 不需要。对于 <kc 作为 OBCN $_{F}$，BCNF 模式的排序为：(R5, D5) <kRc (R2, D2) <kRc （R1, D1），但生成 R1 的函数依赖 EST → V 不是冗余的，生成 R2 的函数依赖 ET → M 或 MT → E 也不是冗余的，所以模式 （R1, D1） 和 （R2, D2） 被添加到分解中。对于生成 R5 的函数依赖 SV → M 也是如此，因此模式 （R5, D5） 也被添加。由于 R5 ⊆ R3，模式 （R5, D5） 从分解中移除。最终分解 D1 = {(R1, D1), (R2, D2), (R3, D3)} 是 <O-3NF/BCNFD-最优的。

以 $>_{k_c}$ 代替 $O'_{3N\,F}$ 作为输入，关键模式按以下顺序排列：$(R_4,\ D_4) <_{kR_c} (R_3,\ D_3) =_{kR_c}$ （R6, D6），生成 R3 的函数依赖 EMS → V 是冗余的，因此被移除，生成 R6 的函数依赖 MST → V 也是冗余的，同样被移除。然而，生成 R4 的函数依赖 EMS → T 并非冗余，模式 （R4, D4） 被添加到分解中。基于 <kc 对于 OBCN′ $_{F}$ ，

BCNF 模式仍按之前的顺序排列：(R5, D5) <kRc (R2, D2) <kRc （R1, D1），但生成 R1 的函数依赖 EST →V 并非冗余，生成 R2 的函数依赖 ET →M 或 MT →E 也并非冗余，因此 （R1, D1） 和 （R2, D2） 被添加到分解中。对于生成 R5 的函数依赖 SV →M 也是如此，所以 （R5, D5） 也被添加进来。由于 R2 ⊆R4，（R2, D2） 被移除。最终的分解 $D_2 = \{(R_1, D_1), (R_4, D_4), (R_5, D_5)\}$ 是 $<_{O\text{-}3NF/BCNF}$ -最优的。

5 实验

通过实验，我们试图回答以下问题。

（E1）主键和非主键的函数依赖关系如何影响性能？（E2）我们能将最先进的算法改进多少？（E3）我们能节省多少更新开销?

对（E1）的回答将更有力地推动我们的研究，并进一步为我们的策略报告提供信息。（E2）侧重于从逻辑层面评估我们的规范化算法相对于先前工作的有效性，包括它们是否达到设计目标以及所需的时间。（E3）则研究我们的优化措施从逻辑层面到操作层面的转化效果。特别是，我们将看到哪种策略在降低完整性维护的开销方面效果最佳。

5.1 实验装置

我们的算法是在 Java 17.0.7 版本中实现的，并在一台 12 代英特尔酷睿 i7-12700 处理器（2.10GHz）、配备 128GB 内存、1TB 固态硬盘以及 Windows 10 操作系统的计算机上运行。我们使用的是 MySQL 8.0.29 数据库。

我们从 12 个数据集（包括现实世界的数据和用于 TPC-H 的基准合成数据）中挖掘数据。这些模式不仅作为函数依赖（FDs）用于数据依赖关系的分析[29, 30, 38]，还在我们之前的工作中用于实验[42]。因此，我们不仅能将我们的算法与最先进的算法进行比较，还能在具有数十、数百甚至数千个函数依赖的实例上对其进行分析，以测试其可扩展性。

算法。我们实现了新的算法，并使用了函数依赖（FD）的挖掘[38]和阿姆斯特朗关系的生成[27]。与之前的工作一样，我们使用了键集和函数依赖集的基数。我们将分别用 iConf-fk （A1） 和 iConf-f （A2） 表示我们的算法 1，其中 O-3NF 分别为（<fc，>kc ）和 <fc，并且在两种情况下 O-BCNF 均为 <kc 。我们将把这些算法的性能与我们之前工作的实现进行比较：Conf (A3) [42]，其在 O-BCNF 中使用 <kc，BC-Cover (A4) [28] 以及 Synthesis (A5) [7]。

5.2 键和主键如何影响性能?

我们研究了添加键或函数依赖（FD）如何影响 TPC-H 基准测试（缩放因子 0.1）的性能，该测试包含 22 个查询、7 个刷新和 3 个插入（分别添加 1000、2000 和 3000 条记录）操作。我们从每个表中挖掘出了五个最合理的最小键和五个最合理的非键函数依赖[38]。对于每个表，键被声明为唯一约束，函数依赖通过触发器实现。图 6a 和 6c 中的每个条形图以百分比形式显示了在添加 f = 1，...，4 个函数依赖后，与仅添加 1 个函数依赖时相比，每种操作的相对速度，平均值基于 30 次运行得出，其中 k = 1，...，5 个键存在。图 6b 和 6d 则将键和函数依赖的角色互换，因此在每个存在 f = 1，...，5 个函数依赖和 1 个键的情况下，我们分别添加 k = 1，...，4 个键。

图 6a 显示了 264%。当添加 FD 到 k 个键（分别为 3、4、5 个最小键）时，更新性能和开销会如何变化。无论添加多少非键 FD，刷新和约束集的平均开销都超过 6400%，实际上，当只有两个键存在时，就会有 FD。

[1] hpi.de/naumann/projects/repeatability/data-profiling/fds.html

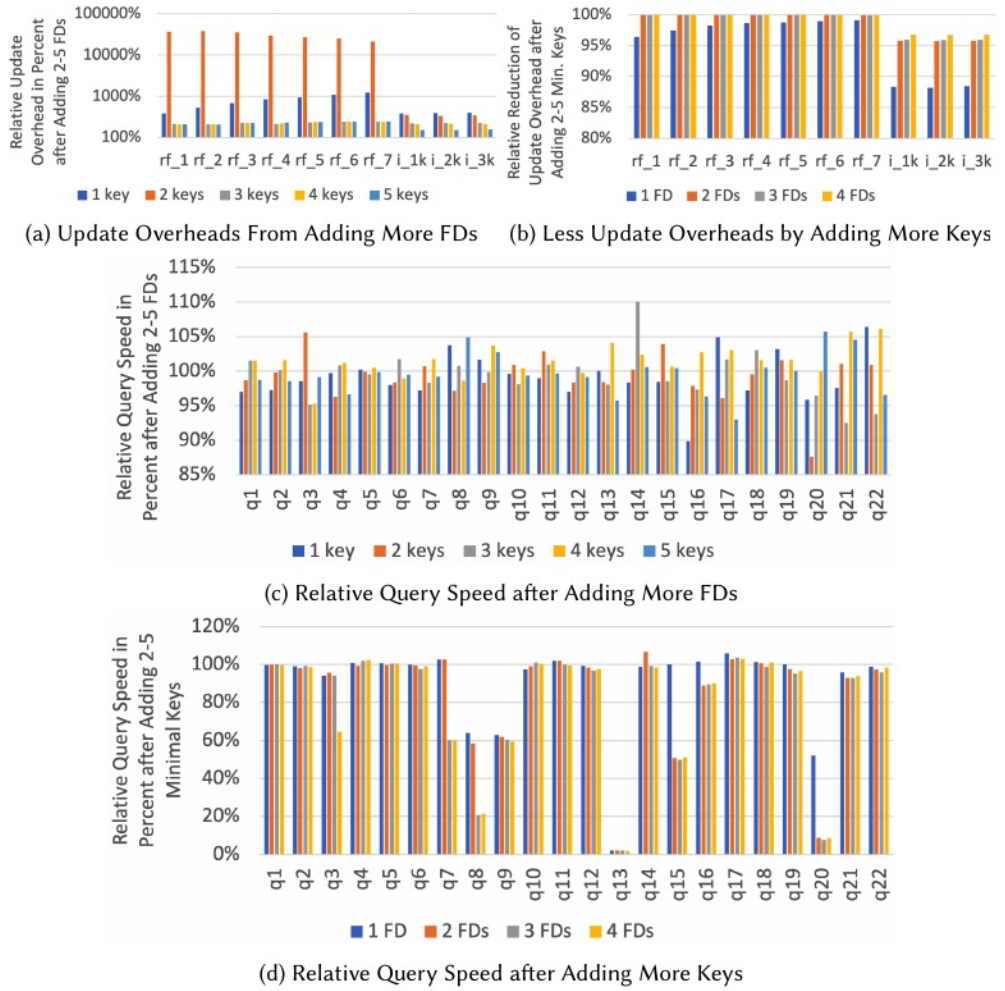

图 6. 更多外键和更多键对 TPC-H 上更新和查询性能的影响

其验证时间并未因键的唯一索引而得到改善，但当存在三个键时，FD 验证时间的扩展性良好。

图 6b 展示了在多个非键依赖关系中添加少量键如何降低更新开销。在刷新和约束集方面，平均减少幅度超过 99.4%，在插入方面，平均减少幅度超过 94%。存在更多的依赖关系会带来巨大的更新开销，但添加一些最小键则能很好地扩展完整性维护。

图 6c 展示了向多个最小键添加函数依赖（FD）如何影响查询性能。22 个查询的平均速度略低于 99.8%。因此，尽管某些查询受到影响，但总体而言，函数依赖对查询性能的影响很小。实际上，总查询时间的差异很小：对于相同的键集，不同函数依赖数量下的最小和最大查询运行时间平均相差约 5%；而更新运行时间平均相差约 75%。因此，图 6c 中观察到的波动是系统原因造成的，并非由非键函数依赖引起。

表 2. 数据集及各数据集的运行时间（单位：毫秒）

	特性			算法运行时间（单位：毫秒）				
数据集	#R	#C	#FD	合成	BC	封面	配置 -	配置 - 配置 - 配置
鲍鱼	4,177	9	137	3	9	11	17	53
成人	四万八千八百四十二		14	2	4	5	5	6
乳房	六百九十九	十一	46	1	1	2	2	3
桥梁	一百零八	十三	142	4	9	11	12	13
回声	一百三十二	七三	527	18	77	93	94	105
肝炎	155	20	8250	2064	11797	13134	14551	15,865
信件	两万零十七		61	4	6	7	7	8
明细项目	6001215	16	3984	2,698	21,269	23,056	23,696	17,364
nc选民	一千零十九		758	115	489	547	595	640
pdbx	一千七百三十五万七百九十九		13	1	4	4	4	4
UniProt	512,000	30	3,703	36,238	213,958	266,825	468,059	266,257
天气	二十六万二千九百一十八		918	4,796	18824	20925	23184	25,140

图 6d 展示了在给定数量的非键约束函数依赖（FD）中添加最少的键对查询性能的影响。平均速度略低于 83.6%，因此速度提高了超过 14.4%。这证实了我们的预期，即由键生成的唯一索引确实提高了查询速度。存在约束函数依赖对查询性能影响不大，但添加最少的键及其唯一索引确实有显著影响。

总之，我们提出了一种框架，将非关键字段依赖（E1）与最小键引入并分离，从而更贴近于 TPC-H 参数的访问和性能需求，使模式设计与操作层面的性能更紧密地结合。实验表明：（1）非关键字段依赖的数量是一个值得最小化的参数；（2）最小键的数量是一个影响更新和查询性能的重要参数；（3）仅依靠字段依赖来维护完整性是不可行的。

5.3 我们的算法有多好?

所有算法均返回无损且依赖关系保持的（LD-）3NF 分解。如果存在 LD-BCNF 分解，BC-Cover 算法会找到一个。Conf 算法通过返回可能的最小 k 值的 k-CONF LD 分解来改进 BC-Cover 算法。iConf-f 算法保证当 O-3NF 小于 fc 且 O-BCNF 小于 kc 时，3NF 的 LD 分解是优化的。iConf-f 算法保证在所有输出模式中，非主键 FD 的最大数量被最小化；如果存在 LD-BCNF 分解（即该最大数量为零），则 iConf-f 算法返回与 Conf 算法相同的结果。iConf-fk 算法通过优先考虑具有更多最小键的模式来打破具有相同数量非主键 FD 的 3NF 模式之间的剩余平局。稍后我们将讨论其他变体，其中 O-3NF 大于 kc 且小于 fc 或小于 fc 且小于 kc 。

5.3.1 运行时分析。表 2 的一部分列出了 12 个数据集的名称、行数（♯R）、列数（♯C）以及数据集所展示的函数依赖集的原子覆盖中的函数依赖数量（♯FD）。与之前的研究一致，我们将两个缺失值视为匹配。若将它们视为不匹配，则会得到不同的函数依赖，但总体观察结果不会改变。

在处理（E2）时，表 2 报告了算法 1 的总运行时间以及之前提到的各步骤所花费的时间，针对每个数据集。由于我们的数据集具有不同的特征，运行时间也存在显著差异。首先，尽管输入规模较大，但所有算法均运行高效。实际上，运行时间最长的是 uniprot 数据集，除了个别情况外，均在 5 分钟以内。

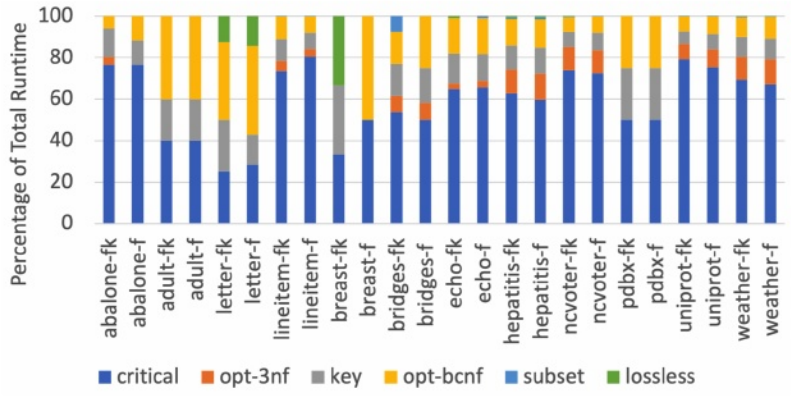

图 7. （A1 - 5）在总运行时间中所占百分比的细分情况

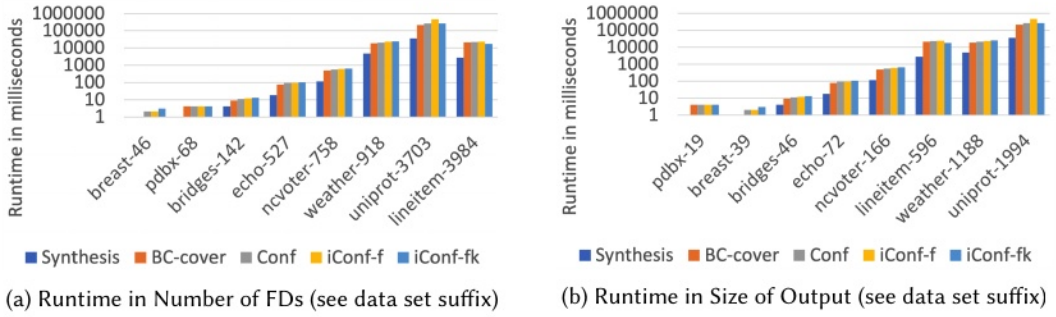

图 8. 运行时间与输入大小和输出大小的关系

iConf-f 的运行时间不到 8 分钟。所有其他 FD 集的运行时间均未超过 30 秒。Synthesis 与其他算法的运行时间存在数量级的差异，这表明 Synthesis 没有尝试将模式分解为 BCNF。不出所料，优化会带来运行时间的开销，特别是 iConf-f 相对于 Conf 的开销。虽然这种差异大多不显著，但在 hepatitis 数据集上不超过 1.5 秒，在 lineitem 数据集上不超过 1 秒，在 weather 数据集上不超过 3 秒，在最极端的 uniprot 数据集上也不超过 3.5 分钟。模式设计是一项关键任务，我们的实验量化了针对目标策略优化模式设计算法所导致的运行时间开销。

图 7 展示了算法的每一步骤对整体运行时间的贡献百分比。由于计算关键模式、在模式上投影的函数依赖的最小覆盖以及查找所有最小键，关键步骤消耗了大部分时间。由于函数依赖和键的数量众多，对 3NF 和 BCNF 的优化也消耗了大量时间，同样，在 BCNF 模式上计算所有最小键的键步骤也是如此。

图 8 分别量化了所有算法的运行时间与输入 FD 数量以及输出模式数量之间预期的指数依赖关系。

5.3.2 输出分析。对于（E3），表 3 报告了将（A1）-（A5）应用于从 12 个数据集挖掘出的函数依赖的结果。对于每个数据集，我们列出了报告结果的算法（具有相同结果的算法合并列出）、输出中关系模式的总数 Size，分为 BCNF 模式的数量 BCNF 和关键模式的数量 3NF，以及平均值

表 3. 基于从数据集中挖掘出的 FD 集的输出分解属性

		分解			BCNF模式		符合第三范式的模式	
数据集	算法	大小	BCNF	3NF	主键数量	分布 #FDs		分配
鲍鱼	A1	26	22	4	1.64	[3:1, 2:12, 1:9]	2	[4:1, 2:1, 1:2]
	A2	23	18	5	1.61	[2:11,1:7]	1.8	[4:1, 2:1, 1:3]
	A3	21	16	5	1.81	[3 比 2，2 比 9，1 比 5]	2.4	[4:2, 2:1, 1:2]
	A4	20	15	5	2.07	[5:1, 3:2, 2:8, 1:4]	2.4	[4:2, 2:1, 1:2]
	A5	21	14	7	1.93	[3:2, 2:9, 1:3]	2.29	[4:2, 3:1, 2:1, 1:3]
成人	A1-5	46	46	0	1.02	[2:1, 1:45]		
乳房	A1-5	39	37	2	1.03	[2:1, 1:36]	1	[1:2]
桥梁	A1-3	46	39	7	1.15	[3 分 1 秒，2 分 4 秒，1 分 34 秒]	1.14	[1:7]
	A4	44	37	7	1.19	[3 分 1 秒，2 分 5 秒，1 分 31 秒]	1.14	[1:7]
	A5	43	34	9	1.21	[3 比 1，2 比 5，1 比 28]	1	[1:9]
回声	A1-3	72	65	7	1.46	[3 分 6 秒，2 分 18 秒，1 分 41 秒]	1.14	[2:1,1:6]
	A4-5	72	65	7	1.58	[4:1, 3:9, 2:17, 1:38]	1.14	[2:1,1:6]
甲型肝炎 1 - 2		1150	842	308	1.12	[3:9, 2:82, 1:751]		1.88 [10:1, 9:1, 8:1, 7:3, 6:4, 5:12, 4:12, 3:31, 2:63, 1:180]
	A3	1130	826	304	1.12	[3:10, 2:82, 1:734]		1.97 [10:1, 9:1, 8:3, 7:4, 6:5, 5:12, 4:13, 3:27, 2:66, 1:172]
	A4	1123	819	304	1.13	[4:1, 3:10, 2:80, 1:728]		1.97 [10:1, 9:1, 8:3, 7:4, 6:5, 5:12, 4:13, 3:27, 2:66, 1:172]
	A5	1113	784	329	1.1	[4:1, 3:6, 2:66, 1:711]		1.93 [10:1, 9:1, 8:3, 7:4, 6:6, 5:13, 4:14, 3:27, 2:67, 1:193]
信件	A1-5	62	62	0	1	[1:62]		
明细项目	A1	590	560	30		1.39 [15:1, 10:1, 6:3, 5:3, 4:5, 3:23, 2:105, 1:419]	1.4	[4:1, 2:9, 1:20]
	A2	596	567	29		1.39 [15:1, 10:1, 6:4, 5:3, 4:5, 3:23, 2:105, 1:425]	1.41	[4:1, 2:9, 1:19]
	A3	587	558	29		1.38 [15:1, 10:1, 6:3, 5:3, 4:5, 3:22, 2:104, 1:419]	2.62	[10:2, 9:1, 5:1, 4:1, 3:2, 2:10, 1:12]
	A4	562	533	29	2.32	[15:1, 11:1, 10:2, 9:9, 8:9, 7 分 19 秒，6 分 26 秒，5 分 26 秒，4 分 26 秒，	2.62	[10:2, 9:1, 5:1, 4:1, 3:2, 2:10, 1:12]
	A5	531	466	65	2.29	[11:1, 10:1, 9:8, 8:9, 7:19, 6:21, 5:25, 4:24, 3:18, 2:30, 1:310]	3 分 26 秒，2 分 46 秒，1 分 34 秒 2.28	[10:3, 9:1, 5:1, 4:4, 3:6, 2:20, 1:30]
nc选民	A1	166	145	21	1.19	[2:27,1:118]	1.19	[3 比 1，2 比 2，1 比 18]
	A2	166	145	21	1.2	[3:1,2:27,1:117]	1.19	[3 比 1，2 比 2，1 比 18]
	A3	168	147	21	1.2	[3:1, 2:28, 1:118]	1.29	[4:1, 2:3, 1:17]
	A4	162	141	21	1.28	[4 分 2 秒，3 分 5 秒，2 分 23 秒，	1.29	[4:1, 2:3, 1:17]
	A5	154	123	31	1.24	[4:1, 3:3, 2:20, 1:99]	1 分 111 秒] 1.35	[4:1, 2:8, 1:22]
pdbx	A1-3	19	14	5	1.21	[2:3,1:11]	1	[1:5]
	A4-5	18	13	5	1.31	[2:4,1:9]	1	[1:5]
UniProt	A1	1992	1576	416	1.13	[4:1, 3:5, 2:187, 1:1383]	1.48	[14:1, 8:1, 7:2, 5:1, 4:13, 3:17, 2:91, 1:290]
	A2	1994	1578	416	1.13	[4:1, 3:5, 2:187, 1:1385]	1.48	[14:1, 8:1, 7:2, 5:1, 4:13, 3:17, 2:91, 1:290]
	A3	1981	1564	417	1.13	[4:1, 3:5, 2:186, 1:1372]	1.56	[14:1, 11:1, 8:1, 7:2, 5:3, 4:17, 3:19, 2:91, 1:282]
	A4	1946	1529	417	1.16	[5:1, 4:2, 3:10, 2:207, 1:1309]	1.56	[14:1, 11:1, 8:1, 7:2, 5:3, 4:17, 3:19, 2:91, 1:282]
	A5	1923	1443	480	1.13	[5:1, 4:1, 3:8, 2:169, 1:1264]	1.53	[14:1, 11:1, 8:1, 7:2, 5:3, 4:17, 3:23, 2:103, 1:329]
天气	A1	1186	796	390	1.2	[6:1, 5:1, 4:3, 3:11, 2:120, 1:660]	2.47	[7:5, 6:10, 5:27, 4:46, 3:68, 2:112, 1:122]
	A2	1188	796	392	1.2	[6:1, 5:1, 4:3, 3:11, 2:119, 1:661]	2.46	[7:5, 6:10, 5:27, 4:46, 3:68, 2:112, 1:124]
	A3	1162	770	392	1.21	[6:1, 5:1, 4:3, 3:11, 2:119, 1:635]	2.56	[9:1,7:7,6:11,5:27,4:50,3:67,2:115,1:114]
	A4	1154	762	392	1.25	[6:2, 5:3, 4:3, 3:16, 2:126, 1:612]	2.56	[9:1,7:7,6:11,5:27,4:50,3:67,2:115,1:114]
	A5	1127	702	425	1.19	[4:1, 3:12, 2:104, 1:585]	2.53	[9:1, 7:8, 6:12, 5:27, 4:53, 3:74, 2:120, 1:130]

在 BCNF（3NF）模式下的最小键（非键属性依赖关系）数量 #键（非键属性依赖关系）分布情况。在分布中，ni 表示恰好具有 si 个最小键（非键属性依赖关系）的 BCNF（3NF）模式的数量。

按设计，BC-Cover（A4）生成的 3NF 模式的临界模式数量从不比 Synthesis（A5）多，通常还更少。按设计，Conf 和 BC-Cover 在 3NF 中生成相同的模式，但 Conf 会针对 <kc 优化 BCNF 模式。实际上，在 abalone、echo、hepatitis、lineitem、ncvoter 和 uniprot 数据集上，Conf 生成的分解具有比 BC-Cover 更好的 D 排名；而在 bridges、pdbx 和 weather 数据集上，它们在最低排名处生成的模式数量有所不同，Conf 生成的模式数量更少。

（A2）作为我们主要分解目标的 iConf-f 表明，（A2）优化的 D 比（A3）Conf 生成的 3NF 分布效果更好。在 ncvoter、uniprot、lineitem 这些数据集上，（A2）消除了包含 10、9 和 5 个函数依赖的 3NF 模式。同样，在 abalone 和 hepatitis 数据集上，（A2）在最低等级上生成的模式也更少。

优化的 iConf-fk（A1）保留了那些通过 iConf-f（A2）关联但具有更少主键的冗余 3NF 模式。与（A2）相比，（A1）消除了更多冗余的 3NF 模式，例如在鲍鱼和天气数据上。对于 ncvoter，3NF 分布相同但模式不同，这使得能够消除具有最少主键的 BCNF 模式。

表 4.算法间平均开销的减少量

比较	总计	按模式
iConf-f 胜过 Conf	20.0%	23.5%
关于 BC 封面的争议	3.0%	6.2%
BC 覆盖合成	5.7%	8.7%

与（A2）和（A3）相比，然而在行项目上，（A1）在（A2）的基础上使用了额外的 3NF 模式，但在某些等级上 BCNF 模式的数量更少。
结论。我们的算法展示了其在优化逻辑模式状态方面的优势，旨在实现诸如最小化关键模式中的非键外键依赖等目标。考虑到计算效率方面的障碍，我们的算法在实际应用中能够高效地达成目标。

5.4 我们能节省多少管理费用?

现在我们将研究在逻辑层面所做的优化如何在操作层面转化为完整性维护。为此，我们将 20k 和 30k 条记录插入到 abalone、hepatitis、lineitem、ncvoter 和 weather 中，这些记录是针对由 iConf-f、Conf、BC-Cover 和 Synthesis 分解所得输出模式的投影。操作重复 10 次，并报告平均运行时间。我们报告了所有约束均由函数依赖（FD）强制执行的总时间以及将函数依赖分为非主键函数依赖和最小键的情况下的总时间。

首先，在统一使用函数依赖（FD）与结合使用非主键函数依赖和最小键时，存在时间单位数量级的差异（小时与分钟，或分钟与秒）。实际上，非主键函数依赖需要触发器，而最小键则由唯一索引支持。这进一步促使我们构建参数化框架，该框架内在地将模式层面的约束表示与操作层面的性能联系起来。

其次，我们的解决方案确实解决了更新效率低下的瓶颈问题。通过减少非主键函数依赖，我们降低了更新开销，其规模超过了以往工作的优化效果。表 4 对此进行了量化，该表比较了在将函数依赖分为非主键函数依赖和最小主键时，各算法在降低更新开销方面的能力。我们报告了在所有更新操作和数据集上，两种算法在总体和每个模式上的平均降低百分比。

图 9 中的各场景细节更新虽不大，但意义重大。在所有 5 个数据集上，包括仅使用函数依赖的最小化缩减覆盖以及将所有最小键与所有非键函数依赖的最小化缩减覆盖相结合的约束集的总时间开销。实际上，iConf-f 在所有不同模式、约束和记录输入大小的场景中均优于此前的最佳算法 Conf。因此，这些优化确实从逻辑层面转化到了操作层面。缩减的幅度有所不同。

优化。我们可能会使用次要参数来打破在使用主要参数后仍然存在的某些关联。表 5 列出了在某些数据集上应用这些策略后分解的属性，这些数据集在这些策略下存在差异。我们注意到差异很小，有时可能会消除或添加少量模式。图 10 展示了这些分解的更新性能。与逻辑层面的小差异一致，在操作层面也存在小差异。总体而言，通过最大化最小键的数量来进一步打破关联，即策略（A1），似乎能进一步小幅降低更新开销。

瓶颈结论：通过在操作努力层面进行归一化处理，我们的框架在更大规模上实现了非关键函数依赖的最小化，从而解决了优化问题，这比之前的工作更进一步。无需将非关键函数依赖与关键函数依赖分开。

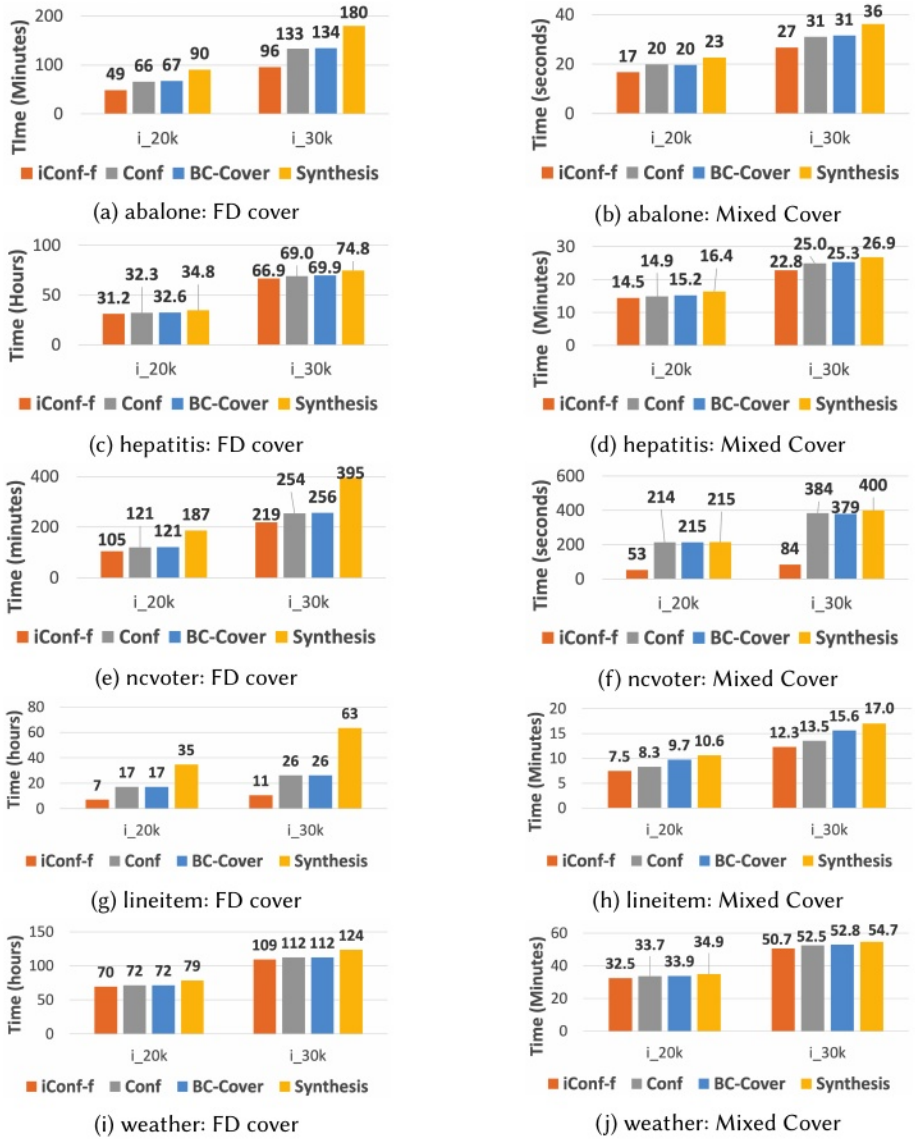

图 9. 在通过不同规范化方法获得的模式上插入 20k 和 30k 条记录时，使用 FD 和混合覆盖维护完整性的开销

最小化键值时，完整性维护性能会大幅下降。选择具有较少函数依赖（FD）的冗余关键模式（如果存在并列情况则选择更多键值）能够最大程度地降低更新开销，这是我们能够证明的最大降幅。

6 结论与未来工作

我们将简要总结关键点以及如何通过将函数依赖集划分为其子集来最小化我们的贡献，从而解决研究中剩余的非关键问题。首先，我们展示了如何通过将一组函数依赖划分为其子集来分离 3NF 模式。

表 5. 优化策略设计的属性

数据	算法	尺寸	BC 第三范式		BCNF 分布	三范式分布
鲍鱼	iConf-去他的	26	22	4	[3:1, 2:12, 1:9]	[4:1, 2:1, 1:2]
	iConf-f<k	23	18	5	[2:11,1:7]	[4:1, 2:1, 1:3]
	iConf 转换为 kf	26	22	4	[3:1, 2:12, 1:9]	[4:1, 2:1, 1:2]
	iConf-f	23	18	5	[2:11,1:7]	[4:1, 2:1, 1:3]
nc选民	iConf-去死	166	145	21	[2:27,1:118]	[3:1, 2:2, 1:18]
	iConf-f < k	164	143	21	[3:1, 2:28, 1:114]	[3:1, 2:2, 1:18]
	iConf 转换为 kf	163	142	21	[2:28 1:114]	[5:1, 3:1, 2:3, 1:16]
	iConf-f	166	145	21	[3:1, 2:27, 1:117]	[3:1, 2:2, 1:18]
明细项目	iConf-去死	590	560		30 [15:1, 10:1, 6:3, 5:3, 4:5, 3:23, 2:105, 1:419]	[4:1, 2:9, 1:20]
	iConf-f < k	602	573		29 [15:1, 10:1, 6:4, 5:3, 4:6, 3:23, 2:106, 1:429]	[4:1, 2:9, 1:19]
	iConf -> kf	558	528	30	[十五比一，十比一，六比三，五比三，四比五，三比二十一，二比九十五，一比三百九十九] [十比一，八比二，六比一，五比一，四比一，三比二，二比九，一比十三]	
	iConf-f	596	567		29 [15:1, 10:1, 6:4, 5:3, 4:5, 3:23, 2:105, 1:425]	[4:1, 2:9, 1:19]

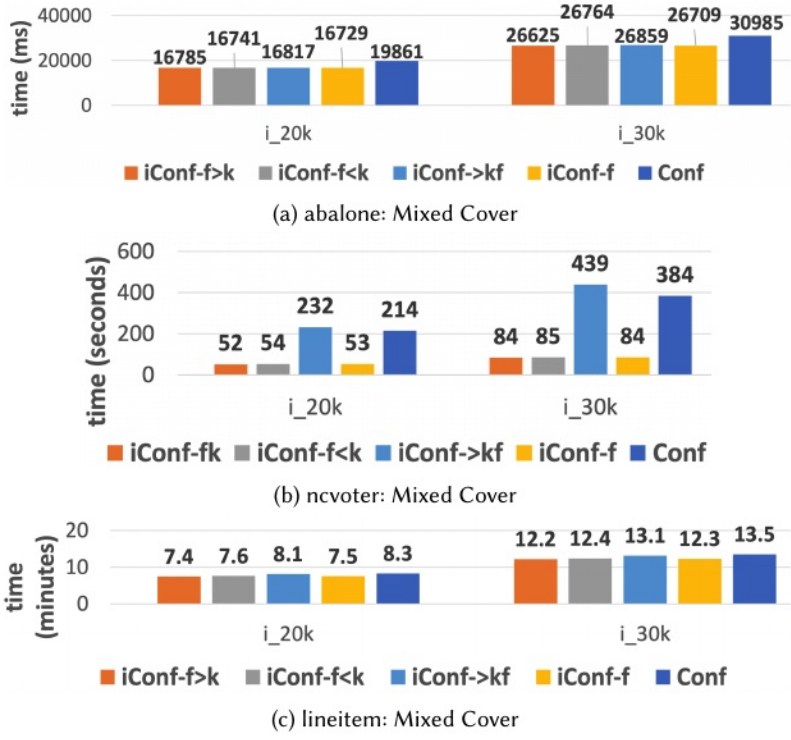

图 10. 插入开销的优化：混合覆盖下的总时间

能够访问这些参数使我们能够基于我们认为在 k 和 f 方面更优的情况来比较 3NF 模式，从而解决（Q1）。虽然 3NF 表明所有完整性约束都可以通过最小键和主属性函数依赖（非键函数依赖，其右侧为一个主属性）来强制执行，但我们定义了（k，f）-3NF，表示所有完整性约束都可以通过 k 个最小键和 f 个主属性函数依赖来强制执行。BCNF 和 k-CONF 被包含在 f = 0 的特殊情况中。这回答了（Q2）。3NF 合成可以根据我们以 k 和 f 为标准声明的任何目标策略进行优化。我们可以通过最小化或最大化参数从各种策略中进行选择。

声明主参数和次参数，并将不同策略合并用于 BCNF 和关键模式。因此，我们通过证明优化针对所声明的目标策略的 3NF 合成来解决（Q3）。尽管一般情况下底层计算问题可能难以解决，但我们的算法在实践中表现高效，尤其是考虑到模式设计的频率远低于操作层面的频繁更新。实际上，我们的参数化框架与操作性能有着内在的联系。在解决（Q4）和完整性维护的瓶颈问题时，我们可以简单地将任何使 f 最小化的目标策略声明为主参数。我们的实验表明，这种策略带来的模式设计在更新性能方面的提升显著优于仅涉及最小键的先前优化所得到的设计。

未来的工作将研究使用键的大小和函数依赖（FD）[25]而非其数量来确定最优覆盖。这里，最优覆盖的昂贵计算与额外性能提升之间的权衡将是一个有趣的分析方向。在数据库投入运行且有关更新和查询的工作负载模式的信息可用之后，对模式进行优化也将是一个有趣的研究课题。此类知识并非经典规范化（包括 BCNF 和 3NF）的输入。还将研究更高范式[8]，如 4NF [10]、5NF [36] 和包含依赖范式[21]。

致谢

本研究由新西兰皇家学会 Te Ap¯arangi 管理的政府资助的马尔登基金理事会资助，资助编号为 MFP-UOA2420。

参考文献

[1] Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Alastair Green, Jan Hidders, Bei Li, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Stefan Plantikow, Ognjen Savkovic, Michael Schmidt, Juan Sequeda, Slawek Staworko, Dominik Tomaszuk, Hannes Voigt, Domagoj Vrgoc, Mingxi Wu, and Dusan Zivkovic. 2023. PG-Schema: Schemas for Property Graphs. Proc. ACM Manag. Data 1, 2 (2023), 198:1−198:25.

[2] Marcelo Arenas. 2006. Normalization theory for XML. SIGMOD Rec. 35, 4 (2006), 57−64.

[3] William Ward Armstrong. 1974. Dependency Structures of Data Base Relationships. In Information Processing, Proceedings of the 6th IFIP Congress 1974, Stockholm, Sweden, August 5-10, 1974. 580−583.

[4] Catriel Beeri and Philip A. Bernstein. 1979. Computational Problems Related to the Design of Normal Form Relational Schemas. ACM Trans. Database Syst. 4, 1 (1979), 30−59.

[5] Catriel Beeri, Philip A. Bernstein, and Nathan Goodman. 1978. A Sophisticate’s Introduction to Database Normalization Theory. In VLDB, September 13-15, 1978, West Berlin, Germany. 113−124.

[6] Philip A. Bernstein. 1976. Synthesizing Third Normal Form Relations from Functional Dependencies. ACM Trans. Database Syst. 1, 4 (1976), 277−298.

[7] Joachim Biskup, Umeshwar Dayal, and Philip A. Bernstein. 1979. Synthesizing Independent Database Schemas. In ACM SIGMOD International Conference on Management of Data, Boston, Massachusetts, USA, May 30 - June 1. 143−151.

[8] C. J. Date and Ronald Fagin. 1992. Simple Conditions for Guaranteeing Higher Normal Forms in Relational Databases. ACM Trans. Database Syst. 17, 3 (1992), 465−476.

[9] Michael DiScala and Daniel J. Abadi. 2016. Automatic Generation of Normalized Relational Schemas from Nested Key-Value Data. In Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016. 295−310.

[10] Ronald Fagin. 1977. Multivalued Dependencies and a New Normal Form for Relational Databases. ACM Trans. Database Syst. 2, 3 (1977), 262−278.

[11] Marie Fischer, Paul Roessler, Paul Sieben, Janina Adamcic, Christoph Kirchherr, Tobias Straeubig, Youri Kaminsky, and Felix Naumann. 2023. BCNF* - From Normalized- to Star-Schemas and Back Again. In Companion of the 2023 International Conference on Management of Data, SIGMOD/PODS 2023, Seattle, WA, USA, June 18-23, 2023. 103−106.

[12] Dimitrios Gunopulos, Roni Khardon, Heikki Mannila, Sanjeev Saluja, Hannu Toivonen, and Ram Sewak Sharm. 2003. Discovering all most specific sentences. ACM Trans. Database Syst. 28, 2 (2003), 140−174.

[13] Christian S. Jensen, Richard T. Snodgrass, and Michael D. Soo. 1996. Extending Existing Dependency Theory to Temporal Databases. IEEE Trans. Knowl. Data Eng. 8, 4 (1996), 563−582.

[14] Jiann H. Jou and Patrick C. Fischer. 1982. The Complexity of Recognizing 3NF Relation Schemes. Inf. Process. Lett. 14, 4 (1982), 187–190.

[15] Henning Köhler. 2006. Finding Faithful Boyce-Codd Normal Form Decompositions. In Algorithmic Aspects in Information and Management, Second International Conference, AAIM 2006, Hong Kong, China, June 20-22, 2006, Proceedings. 102–113.

[16] Christoph Köhnen, Stefan Klessinger, Jens Zumbrägel, and Stefanie Scherzinger. 2023. A Plaque Test for Redundancies in Relational Data. In Workshops at VLDB, Vancouver, Canada, August 28 - September 1, 2023.

[17] Solmaz Kolahi. 2007. Dependency-preserving normalization of relational and XML data. J. Comput. Syst. Sci. 73, 4 (2007), 636–647.

[18] Carol Helfgott LeDoux and Douglas Stott Parker Jr. 1982. Reflections on Boyce-Codd Normal Form. In Eigth International Conference on Very Large Data Bases, September 8-10, 1982, Mexico City, Mexico, Proceedings. 131–141.

[19] Mark Levene and George Loizou. 1999. Database Design for Incomplete Relations. ACM Trans. Database Syst. 24, 1 (1999), 80–125.

[20] Mark Levene and George Loizou. 1999. A guided tour of relational databases and beyond. Springer.

[21] Mark Levene and Millist W. Vincent. 2000. Justification for Inclusion Dependency Normal Form. IEEE Trans. Knowl. Data Eng. 12, 2 (2000), 281–291.

[22] Sebastian Link and Henri Prade. 2019. Relational schema design for uncertain data. Inf. Syst. 84 (2019), 88–110.

[23] Sebastian Link and Ziheng Wei. 2021. Logical Schema Design that Quantifies Update Inefficiency and Join Efficiency. In SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021. 1169–1181. [24] Claudio L. Lucchesi and Sylvia L. Osborn. 1978. Candidate Keys for Relations. J. Comput. Syst. Sci. 17, 2 (1978), 270–279. [25] David Maier. 1980. Minimum Covers in Relational Database Model. J. ACM 27, 4 (1980), 664–674.

[26] David Maier. 1983. The Theory of Relational Databases. Computer Science Press.

[27] Heikki Mannila and Kari-Jouko Räihä. 1986. Design by Example: An Application of Armstrong Relations. J. Comput. Syst. Sci. 33, 2 (1986), 126–141.

[28] Sylvia L. Osborn. 1979. Testing for Existence of a Covering Boyce-Codd Normal Form. Inf. Process. Lett. 8, 1 (1979), 11–14.

[29] Thorsten Papenbrock and Felix Naumann. 2016. A Hybrid Approach to Functional Dependency Discovery. In SIGMOD. 821–833.

[30] Thorsten Papenbrock and Felix Naumann. 2017. Data-driven Schema Normalization. In Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017. 342–353. [31] Avi Silberschatz, Henri F. Korth, and S. Sudarshan. 2019. Database System Concepts (7 ed.). McGraw-Hill.

[32] Philipp Skavantzos and Sebastian Link. 2023. Normalizing Property Graphs. Proc. VLDB Endow. 16, 11 (2023), 3031–3043.

[33] Philipp Skavantzos and Sebastian Link. 2025. Entity/Relationship Graphs: Principled Design, Modeling, and Data Integrity Management of Graph Databases. Proc. ACM Manag. Data 3, 1 (2025), 40:1–40:26.

[34] Philipp Skavantzos and Sebastian Link. 2025. Third and Boyce-Codd normal form for property graphs. VLDB J. 34, 2 (2025), 23.

[35] Don-Min Tsou and Patrick C. Fischer. 1980. Decomposition of a relation scheme into Boyce-Codd Normal Form. In Proceedings of the ACM 1980 Annual Conference, Nashville, Tennessee, USA, October, 27-29, 1980. 411–417.

[36] Millist W. Vincent. 1997. A Corrected 5NF Definition for Relational Database Design. Theor. Comput. Sci. 185, 2 (1997), 379–391.

[37] Millist W. Vincent. 1999. Semantic Foundations of 4NF in Relational Database Design. Acta Informatica 36, 3 (1999), 173–213.

[38] Ziheng Wei and Sebastian Link. 2019. Discovery and Ranking of Functional Dependencies. In 35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019. 1526–1537.

[39] Ziheng Wei and Sebastian Link. 2021. Embedded Functional Dependencies and Data-completeness Tailored Database Design. ACM Trans. Database Syst. 46, 2 (2021), 7:1–7:46.

[40] Cong Yu and H. V. Jagadish. 2008. XML schema refinement through redundancy detection and normalization. VLDB J. 17, 2 (2008), 203–223.

[41] Carlo Zaniolo. 1982. A New Normal Form for the Design of Relational Database Schemata. ACM Trans. Database Syst. 7, 3 (1982), 489–499.

[42] Zhuoxing Zhang, Wu Chen, and Sebastian Link. 2023. Composite Object Normal Forms: Parameterizing Boyce-Codd Normal Form by the Number of Minimal Keys. Proc. ACM Manag. Data 1, 1 (2023), 13:1–13:25.

[43] Zhuoxing Zhang and Sebastian Link. 2024. Mixed Covers of Keys and Functional Dependencies for Maintaining the Integrity of Data under Updates. Proc. VLDB Endow. 17, 7 (2024), 1578–1590.

[44] Zhuoxing Zhang and Sebastian Link. 2025. Mixed covers: optimizing updates and queries using minimal keys and functional dependencies. VLDB J. 34, 3 (2025), 29.

Received October 2024; revised January 2025; accepted February 2025