量化更新低效性和连接高效性的逻辑模式设计

Sebastian Link
s.link@auckland.ac.nz
奥克兰大学
新西兰奥克兰

Ziheng Wei
z.wei@auckland.ac.nz
新西兰奥克兰大学 奥克兰
新西兰

摘要

经典范式化的目标是在更新数据时保持数据一致性，且付出最小的努力。仅考虑函数依赖（FD）的情况下，只有在存在保持 FD 的 Boyce-Codd 范式（BCNF）分解的特殊情况下，这一目标才能实现。我们证明，在其他所有情况下，所需的努力既无法控制也无法量化。对此，我们提出了由正整数 ℓ 参数化的 ℓ 有界基数范式。对于每一个 ℓ，该范式条件要求每个实例中，每个非平凡 FD 左边的每个值组合在不超过 ℓ 个元组中出现。当 $\ell = 1$ 时，即为 BCNF。我们证明，处于这种范式下的模式具有以下特征：i) 没有 ℓ 级数据冗余和更新低效问题；ii) 允许 ℓ 级连接效率。我们还建立了算法，用于计算在所有保持 FD 的分解中可达到的最小 ℓ 值下的 ℓ 有界基数范式模式。另外一些算法 i) 基于部分函数依赖的丢失来实现更低的维护成本，ii) 根据导致高维护成本的优先级函数依赖对模式进行分解。我们的框架在逻辑设计阶段就为反规范化提供信息。特别是，级别 ℓ 量化了物化视图的增量维护和连接支持。通过合成数据和真实世界数据进行的实验说明了由我们的算法生成的模式具有哪些特性，以及这些特性如何预测在没有和有物化视图的情况下对基于这些模式的实例执行更新和查询操作的性能。

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。

关键词

基数约束；数据冗余；函数依赖；连接；范式；规范化；更新

ACM 参考格式：

塞巴斯蒂安·林克和魏子恒。2021 年。量化更新低效性和连接高效性的逻辑模式设计。载于 2021 年会议论文集。

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGMOD ’ 21, June 20–25, 2021, Virtual Event, China

©2021 Copyright held by the owner/author(s). Publication rights licensed to ACM. ACM ISBN 978-1-4503-8343-1/21/06...$15.00 https://doi.org/10.1145/3448016.3459238

2021 年 6 月 20 日至 25 日，中国，线上，第 47 届国际数据管理大会（SIGMOD ’ 21），美国纽约，ACM 出版，13 页。https://doi.org/10.1145/3448016.3459238

1 简介

模式设计旨在找到一种数据布局，以促进常见查询和更新的高效处理。这一问题颇具挑战性，因为数据冗余通常会导致更新效率低下，但能提高连接效率。到目前为止，这一挑战是通过在逻辑模式设计期间进行规范化以实现更新效率，然后在物理设计期间进行反规范化以提高查询效率来解决的。特别是，反规范化仅在数据库投入运行且在规范化数据库上的数据访问模式显现之后才进行。

经典范式化的目标是在更新数据时以最小的努力保持数据的一致性。范式化旨在消除未来数据库实例中任何冗余数据值的出现。这通过将导致数据冗余的函数依赖（FD）结构化地转换为禁止数据冗余的键来实现。著名的博伊斯 - 科德范式（BCNF）要求每个非平凡函数依赖的左侧都必须是键。因此，由函数依赖转换为键的数据冗余永远不会发生。然而，虽然每个模式都可以分解为 BCNF，但在这一过程中可能会丢失函数依赖，从而仍会导致数据冗余[5]。第三范式（3NF）给出了一个更宽松的条件，即每个非平凡函数依赖的左侧必须是键，或者右侧的每个属性都必须是主属性（即属于某个最小键的一部分）。3NF 合成可以将每个模式转换为 3NF 而不丢失任何函数依赖，但某些函数依赖仍会导致数据冗余，因为它们未被转换为键。因此，只有在无需任何努力即可保持数据一致性时，经典范式化才能衡量其成功。只有在存在保持 FD 的 BCNF 分解的情况下，这种情况才有可能。在所有其他情况下，工作量既无法控制也无法衡量。

以事件管理模式 Hap 为例，其中每条记录代表在某个 V（地点）于 T（时间）发生的 E（事件），且由某个 C（公司）负责。Hap 使用函数依赖来建模业务规则。函数依赖 E → C 表示每个事件仅由一家公司负责，V → C 表示同一地点不能由不同公司负责，VT → E 表示同一地点同一时间不会发生不同事件，ET → V 表示同一事件不会在同一时间发生在不同地点，CT → V 表示同一时间一家公司不会负责不同地点。最小键为 ET、VT 和 CT。因此，Hap 中的每个属性都是主属性。虽然 Hap 处于第三范式，但它存在

表 1：处于第三范式的模式哈普 以及其分解 D1 以及实例（冗余数据值加粗）

（a）实例 r 覆盖 Hap

哈普

事件	聚会地点	公司	时间
聚会，派对	$_1$	千（重量单位）$_1$	
..	..	..	..
.	.	.	
聚会，派对	$v\ell_1$	千（重量单位）	$t1l$
$_1$	穹顶	巨型的；巨大的	t_1' 超大的
..	..	..	..
.	.	.	
$e\ell_2$	穹顶	巨型的；巨大的	$t\ell''_2$ 超大的

(b) Hap 的分解 D1 = {R2, R3, R4} 及其投影实例 {ri = πRi (r)}i=2 4

R_2

聚会地点	公司
v_1	千（在计量单位中）
..	..
.	.
$v\ell 1$	千（重量单位）
穹顶	巨型的；巨大的；超大的

R_3

事件	聚会地点	时间
聚会，派对	$_1$	t_1
..	..	..
.	.	.
聚会，派对	$v\ell_1$	$t1l$
穹顶		t_1'
..	..	..
.	.	.
e^2	穹顶	$t\ell'_2$

R_4

事件	公司	时间
聚会，派对	千（重量单位）	t_1
..	..	..
.	.	.
聚会，派对	千（重量单位）	$t1l$
$_1$	巨型的；巨大的	t_1' 超大的
..	..	..
.	.	.
$e\ell_2$	巨型的；巨大的	$t\ell''_2$ 超大的

不存在保持函数依赖的 BCNF 分解。在保持函数依赖的前提下，经典范式化无法实现更多。

然而，我们无法衡量 Hap 维护数据一致性的努力程度。在表 1a 的实例 r 中，由于函数依赖 E→C，C 值为 Kilo 的每个 $\ell 1 > 1$ 次出现都是冗余的；由于函数依赖 V→C，C 值为 Mega 的每个 $\ell 2 > 1$ 次出现也是冗余的。请记住，只要每次将某个值的出现更新为不同值都会导致函数依赖的违反，那么该值的出现就是冗余的。我们将给定数据值可以在其中冗余出现的不同元组的数量称为数据冗余度。显然，Hap 上的数据冗余度是无界的。对于每个处于 3NF 但不在 BCNF 的模式，情况都是如此。正如我们将要展示的，BCNF 分解过程中丢失的每个函数依赖仍然会导致无界的数据冗余度。因此，需要更新的数据值的数量是先验无界的。更改 Kilo 的一次出现意味着总共需要更新 $\ell 1$ 次 Kilo 的出现以确保数据一致性。我们将为实现一致性而需要更新的出现的总数称为更新低效度。因此，对于保持 FD 的 BCNF 分解，数据一致性的努力程度处于最优的 1 级，而在其他所有情况下则是无界的。

然而，在实际应用中，$\ell 1$ 和 $\ell 2$ 表示基数约束（CC）的上限。它们构成了一类不同于函数依赖（FD）的业务规则。对于正整数 ℓ，基数约束 $\text{card}(X) \leqslant \ell$ 表示每个实例在 X 中的所有属性上具有匹配值的不同记录最多为 ℓ 条。当 $\ell = 1$ 时，X 是一个键。当 $\ell = \infty$ 时，未指定上限。例如，基数约束 $\text{card}(E) \leqslant \ell 1$，其中 $\ell 1 = 1000$（1k），表示每个事件最多有 1000 种不同的场地和时间组合。同样，基数约束 $\text{card}(V) \leqslant \ell 2$，其中 $\ell 2 = 1000000$（1m），表示每个场地最多有 100 万种不同的活动和时间组合。基数约束为模式设计提供了超出经典范式的指导。给定基数约束，3NF 模式 Hap 允许 $\ell 2$ 级别的数据冗余和更新效率低下。若没有基数约束，则未指定任何整数上限，除非其左侧为键，否则每个非平凡或丢失的函数依赖都会导致无限制的数据冗余和更新效率低下。因此，我们建议在模式设计中纳入基数约束。

通过 CC，我们可以量化在更新操作下保持数据一致性所需的工作量。例如，我们可能会问，对于 Hap，哪些保持 FD 的分解能将级别 ℓ 最小化。应用我们的新算法之一，可得到表 1b 中的模式 D1 = {(R2, Σ2), (R3, Σ3), (R4, Σ4)}，其中

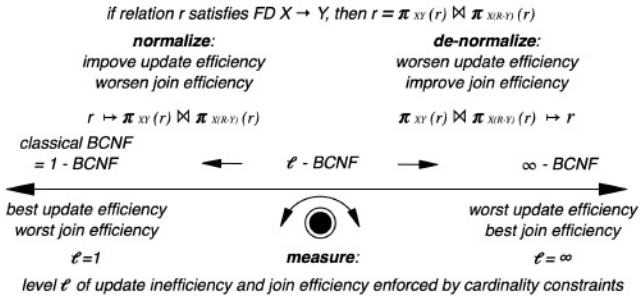


图 1：实现数据一致性所需的努力级别 ℓ 及其对更新和连接效率的影响

- $R_2 = \text{CV}$ 和 $\Sigma_2 = \{V \rightarrow C\}$
- $R_3 = \text{ETV}$ 且 $\Sigma_3 = \{TV \rightarrow E, ET \rightarrow V\}$，并且
- $R_4 = \text{CET}$ 和 $\Sigma_4 = \{CT \rightarrow E, E \rightarrow C\}$.

模式（R2，Σ2）和（R3，Σ3）均处于 BCNF 形式，不存在任何冗余数据值。然而，R4 处于 3NF 形式，但由于 E→C 的存在，仍存在 $\ell 1 = 1k$ 级别的数据冗余和更新效率低下问题。实际上，对于 Hap 的任何保持函数依赖的分解，$\ell 1 = 1k$ 都是所能达到的最佳级别。给定的函数依赖 CT→V 在分解 D1 中得以保留，因为它由 CT→E 和 ET→V 推导得出。分解 D2 = {(R1， Σ1), (R3， Σ3), (R5， Σ5)} 具有

- $R_1 = \text{EC}$ 且 $\Sigma_1 = \{E \rightarrow C\}$，并且
- $R_5 = \text{CTV}$ 且 $\Sigma_5 = \{CT \rightarrow V, V \rightarrow C\}$

处于第三范式，并实现了 $\ell 2 = 1m$ 的数据冗余度。

在陈的开创性论文[9]中，CCs 已经被引入，并且被 UML、XML 和 OWL 所采用。令人惊讶的是，它们尚未用于规范化。我们还注意到，CCs 可以量化模式的连接效率水平。我们将后者定义为在模式的任何实例中，FD 可以与某些冗余值连接的最大值的数量。表 1b 中 R4 上的实例 r4 满足 $E \rightarrow C$。因此，r4 等于 πEC （r4） ▷◁ πET （r4） 的连接。实际上，冗余数据 C 值 Kilo 可以通过 E 值 Party 与 ℓ 1 个 T 值 t1， . . . ， $t\ell 1$ 进行连接。没有 CCs，连接强度是无界的，因此经典理论无法提供模式连接效率的见解。CCs 可以在数据库运行之前就为物化视图的选择提供信息，这里物化视图的级别 ℓ 既量化了其增量更新所需的努力，

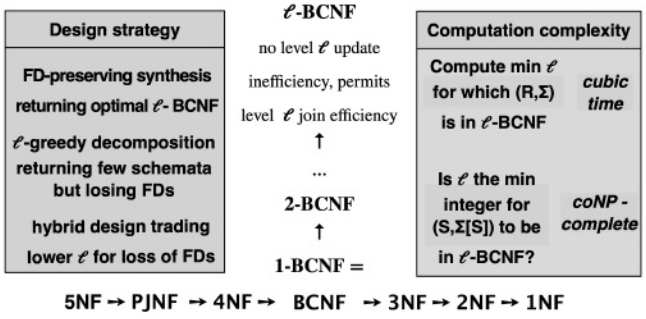


图 2：ℓ-BCNF 的成就

维护 D1，其级别以及维护其支持和连接。连接效率定义为 Hap ℓ 2 = 1m.view 贡献。我们的两个主要贡献是：

我们建立了首个用于逻辑模式规范化的框架，该框架量化了在更新期间实现数据一致性所需的工作量。

我们的框架还对模式的连接效率进行了量化，这意味着它在逻辑设计阶段就已经为反规范化提供了依据。

图 1 的更新展示了我们的主要思路和努力方向，特别是模式的能力。其中，ℓ 进行了量化。图 2 展示了我们的技术成果，具体为：

我们放宽了经典的 BCNF 条件，允许每个非平凡函数依赖左侧的每种值组合在每个实例中最多出现在 ℓ 个元组中。因此，我们得到了 ℓ 有界基数范式（ℓ-BCNF）的无限层次结构，其中经典 BCNF 对应于 ℓ = 1 的情况。因此，能够达到 ℓ-BCNF 的最小 ℓ 值衡量了实现数据一致性所需的努力。

（2）我们证明，对于每一个 ℓ，处于 ℓ-BCNF 范式的模式能够表征出这样的实例：i）不存在 ℓ 级别的数据冗余和更新低效问题；ii）允许达到 ℓ 级别的连接效率。

（3）我们为在保持函数依赖的情况下计算可实现模式的最小级别建立了 ℓ-BCNF 算法。另一种算法根据函数依赖所导致的数据冗余程度对其进行优先级排序，以实现无损分解。该算法生成的输出模式较少，但可能会丢失函数依赖。将这两种算法结合为一种混合策略，我们通过丢失函数依赖进一步降低了从保持函数依赖的分解中可实现的最小级别。

（4）级别捕获 ℓ 同时反映了在更新操作下维护物化视图所付出的增量选择努力，以及视图对连接查询的支持。

通过使用合成数据和真实世界数据进行的实验，说明了由我们的算法生成的模式具有哪些特性，以及这些特性如何预测在没有和有物化视图的情况下基于这些模式的实例的更新和查询操作的物理性能。

组织结构。我们在第 2 节回顾了预备知识。第 3 节介绍了我们的 ℓ-BCNF 家族及其计算特性。第 4 节确定了 ℓ-BCNF 中模式在更新和连接方面的特性。我们在第 5 节讨论了三种设计算法。实验结果在第 6 节给出。相关工作在第 7 节讨论。我们在第 8 节总结。技术报告包含详细信息和数据集链接 [36]。

2 预备事项

函数依赖的经典范式化是大多数关系数据库教材中的内容。表 2 回顾了我们在此处使用的定义和符号。此外，我们还明确回顾了一些不太熟悉的关于多值依赖的概念。有关经典范式化的详细背景知识，我们建议读者参考一些经典文献 [3, 4, 38]。

令 N 趋于无穷大

记 ≥ 1 表示正整数和无穷大。关于关系模式 R 的基数约束（CC）是一个表达式 card(X) ⩽ℓ，其中 X ⊆R 且 ℓ∈N∞⩾1 。若关系 r 满足 card(X) ⩽ℓ，则意味着在 r 中，所有在 X 中的属性值都相同的元组数量不超过 ℓ，即

$$\forall t_1, \ldots, t_{\ell+1} \in r \quad (t_1(X) = \ldots = t_{\ell+1}(X) \Rightarrow \exists i, j \in \{1, \ldots, \ell+1\}(t_i = t_j)) .$$

例如，表 1a 中的关系满足 card(EC) ⩽ ℓ1 = 1k，但由于存在 1k 个不同的元组，其 E 值为 Party 且 C 值为 Kilo，所以违反了 card (EC) ⩽ 999 的条件。

表 3 展示了阿姆斯特朗公理集 A = {R， E, T}，该集合仅对函数依赖形成公理化 [2]；集合 {S， U， L， A} 仅对多值依赖形成公理化 [21]；以及表 3 中所有规则组成的集合 L，该集合对多值依赖和函数依赖共同形成公理化 [21]。在我们的示例中，我们可以应用规则 E 对 E → C 推导出 E → EC。然后，我们应用规则 P 对 E → EC 和 card(EC) ⩽ ℓ1 推导出 card(E) ⩽ ℓ1。

公理化的目标是高效的算法。函数依赖 X→Y 被函数依赖集 Σ 所蕴含，当且仅当 Y 是 X 在 Σ 下的属性集闭包 XΣ+ 的子集。这导致了函数依赖蕴含的线性时间算法[3]。尽管在单独使用时，连接条件和函数依赖的表达能力更强，但它们的组合蕴含可以简化为仅对函数依赖的蕴含，通过将任何包含连接条件和函数依赖的集合 Σ 转换为函数依赖集即可实现。

$$\Sigma[FD] = \{X \to Y \mid X \to Y \in \Sigma\} \cup \{X \to R \mid card(X) \leq 1 \in \Sigma\} \text{ [21] .}$$

本质上，函数依赖 X → Y 由 CC/FD 集合 Σ 推导得出，当且仅当它由函数依赖集合 Σ[FD] 推导得出；CC 卡片(X) ⩽ ℓ 由 CC/FD 集合 Σ 推导得出，当且仅当存在某个卡片(Y) ⩽ ℓ′∈Σ∪{卡片(R) ⩽ 1}，使得 X → Y 由 Σ[FD] 推导得出且 ℓ′⩽ ℓ[21]。

例如，如果 Σ 包含 E → C 且 card(EC) ⩽ 1k，那么 card(E) ⩽ 2k 由 Σ 推出，因为 EΣ[FD] + = EC 且 EC ⊆ EΣ [FD] + 对于 CC 卡（EC）⩽ 1k ∈ Σ 且 ℓ′ = 1k ⩽ 2k = ℓ 。

因此，对于组合的 CC 和 FD 决定蕴含的时间复杂度为 O（|Σ|×||Σ||），其中 |Σ| 表示 Σ 中元素的数量，||Σ|| 表示 Σ 中属性出现的总次数[21]。

3 关于 ℓ-BCNF 的家族

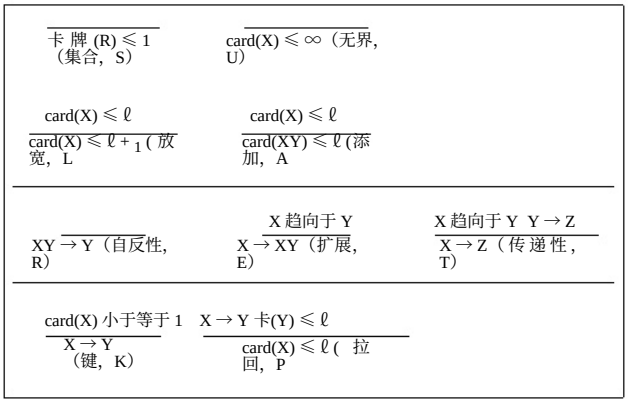
并且我们在第 4 节中研究了 ℓ 有界更新的正常效率形式和连接效率，其基数水平的一些计算性质也被确立。

只要每个非平凡函数依赖 X→Y 的左部 X 都是键，模式就处于 BCNF。由于当 card(X)⩽1 成立时 X 就是键，因此 CC 提供了一种方便的机制来放宽 BCNF 条件，具体如下。

表 2：关系规范化框架中的概念及其定义

概念	定义
关系模式 R	有限属性集 R 属于属性 A，每个属性都有一个可能值的域 dom(A)
数据库模式	有限的关系模式集合 D = {R1, . . . ,Rn}
元组 t 在 R 上	函数 t : R → Ð A∈R dom(A) 且 t(A) ∈ dom(A
关系 r 在 R 上	有限元组集 R，其元素为 R 上的元组
关系 r 在 X 上的投影	r(X) = {t(X) | t ∈ r} 其中 t(X) 是元组 t 在属性集 X 上的投影
r 满足函数依赖 X → Y (r |= X → Y	对于关系 r 中的所有元组 t 和 t ‘，如果 t(X) = t’ (X)，则 t(Y) = t'(Y
r 满足函数依赖集 Σ（|=r Σ）	对于所有 σ∈Σ，|=r σ
Σ 在 R 上蕴含 ϕ（Σ |= ϕ）	对于关系模式 R 上的所有关系 r：若 |=r Σ，则 |=r ϕ
从Σ推导出ϕ（Σ证明R ϕ）	可以通过应用规则集 R 中的规则从 Σ 推导出 ϕ 。
函数依赖集 Σ 的语义闭包	Σ* = {ϕ | Σ |= ϕ}（Σ 所蕴含的所有函数依赖）
函数依赖集 Σ 的语法闭包	Σ+ = {ϕ | Σ ⊠R ϕ} （所有可由Σ通过应用R中的规则推导出的函数依赖） R
在Σ下X的属性集闭包	$X_{\Sigma^+} = \{A \in R \mid X \to A \in \Sigma^+\}$ -> $X_{\Sigma^+} = \{A \in R \mid X \to A \in \Sigma^+\}$（由属性集 X 函数确定的所有属性） R
公理化，记作 R	规则集 R 是可靠的：Σ* ⊆ Σ+R ，并且是完备的：Σ+R ⊆ Σ*
Σ 的 FD 集覆盖	FD 集合 Θ 是 Σ 的覆盖，当且仅当 Θ = Σ（Θ 所蕴含的函数依赖与 Σ 相同）
非冗余（L-约简）函数依赖集 Σ 的规范覆盖 Σ	对于所有 σ∈Σ，Σ−{σ} ⊠|= σ（并且对于所有 X →Y ∈Σ，不存在真包含于 X 的 Z 使得 Σ|= Z →Y） 覆盖 Σ 是无冗余的、L 约简的，并且左部是唯一的
阿姆斯特朗公理，记为 A（（R，Σ）的最小）键	具有自反性公理 R、扩展规则 E 和传递性规则 T 的规则集（见表 3）（最小）X ⊆ R 使得 Σ |= X → R
关系 R 中的主属性 A	属性 A 包含在关系 R 和模式 Σ 的某个最小键中
关系模式 （R， Σ） 属于 BCNF（第三范式）	对于Σ+ 中的每一个非平凡函数依赖 X→Y，X 是 （R， Σ） 的键（或者 Y - X 中的每一个 A 都是主属性）
（无损）分解（R，Σ）	数据库模式 D 与 ∪S ∈D S = R（并且对于所有 R 关系 r：如果 |=r Σ，则 r = ▷◁S ∈D r(S)）
函数依赖集 Σ 在 S 上的投影 Σ[S]	Σ[S] = {X → Y ∈ Σ* | X, Y ⊆ S} （由Σ所蕴含且属性在属性集S中的所有函数依赖）
（R，Σ）的 BCNF（3NF）分解	（R，Σ）的分解 D，使得对于所有 S ∈ D，（S，Σ[S]）处于 BCNF（3NF）状态，且对于所有 X → Y ∈ Σ，都有（S ∈ D，Σ[S]）|= X → Y。
（R，Σ）的保持 FD 分解	

表 3：CC 和 FD 的公理化表述



定义 3.1. 设 Σ 表示关系模式 R 上的一组 CC 和 FD，且设 ℓ∈N∞⩾1 。（R， Σ） 处于 ℓ 有界基数范式（ℓ-BCNF）当且仅当对于所有 X →Y ∈ΣL+ ，其中 Y ⊠X ，我们有 card(X) ⩽ℓ∈ΣL+ 。

例 3.2. 对于我们正在讨论的示例 Hap，考虑函数依赖集 Θ，其中包含 E → C、V → C、VT → E、ET → V 和 CT → V。虽然（Hap，Θ）处于第三范式，但由于满足 card(E) ⩽ ℓ ∈ ΘL+ 的最小 ℓ 为 ℓ = ∞，所以它处于 ∞-BCNF。直观地说，这与我们对数据冗余、更新低效和连接效率无界的理解相符。现在考虑（Hap，Σ），其中 Σ 包含 Θ 中的函数依赖以及基数约束 card(E) ⩽ 1k 和 card(V) ⩽ 1m。由于 VT、ET 和 CT 是（最小）主键，我们得到基数约束 card(VT) ⩽ 1、card(ET) ⩽ 1 和 card(CT) ⩽ 1。因此，Σ 中的每个非平凡 X → Y 都满足 card(X) ⩽ 1m ∈ ΣL+。后一条件等价于处于 1m-BCNF（定理 3.5），所以（Hap，Σ）处于 1m-BCNF。由于存在函数依赖 V → C ∈ Σ，但满足 card(V) ⩽ ℓ2 ∈ ΣL+ 的最小 ℓ2 为 ℓ2 = 1m，所以（Hap，Σ）不在任何 ℓ-BCNF 中，其中 ℓ < 1m。

我们对 ℓ-BCNF 的定义具有在覆盖下不变的优良性质，如下结果所述。

定理 3.3. 设 Σ 和 Θ 表示在关系 R 上的两个 CC/FD 集合，且它们相互覆盖。对于所有 ℓ∈N∞⩾1 ，（R， Σ） 处于 ℓ-BCNF 当且仅当 （R， Θ） 处于 ℓ-BCNF 。

定理 3.3 意味着我们无需担心如何将应用语义表示为完整性约束。例如，设 Σ′ 表示 Θ 中的函数依赖以及基数约束 card(EC) ⩽ 1k 和 card(VC) ⩽ 1m。那么，Σ 和 Σ′ 相互覆盖，这一点从添加规则 A、扩展规则 E 和拉回规则 P 中很容易看出。因此，（Hap，Σ′） 在 1m-BCNF 中，但不在任何 ℓ-BCNF 中，其中 ℓ < 1m。

我们的定义产生了一个具有严格层次结构的语法范式族。处于底层的是 1-BCNF，它等同于经典的博伊斯 - 科德范式。

定理 3.4. 对于每一个模式 （R， Σ） 以及每一个正整数 ℓ，我们有：若 （R， Σ） 属于 ℓ-BCNF，则 （R， Σ） 也属于 ℓ + 1-BCNF。然而，对于每一个正整数 ℓ，都存在属于 ℓ + 1-BCNF 但不属于 ℓ-BCNF 的模式 （R， Σ）。

Algorithm 1 Strongest ℓ-Bounded Cardinality Normal Form

Require: (R, Σ) with CC/FD set Σ over schema R
Ensure: Minimum integer ℓ such that (R, Σ) in ℓ-BCNF
1: **if** $\Sigma_{\text{FD}} = \emptyset$ **then**
2: **return** 1
3: $\ell \leftarrow \infty$
4: $\Sigma_{\text{card}} \leftarrow \Sigma_{\text{card}} \cup \{card(R) \leq 1\}$
5: **for all** $X \rightarrow Y \in \Sigma_{\text{FD}}$ such that $Y \not\subseteq X$ **do**
6: **if** ℓ_X has not been defined before **then**
7: $\ell_X \leftarrow \infty$
8: **for all** $card(Y) \leq \ell' \in \Sigma_{\text{card}}$ with $\ell' < \ell_X$ **do**
9: **if** $Y \subseteq X^+_{\Sigma[\text{FD}]}$ **then**
10: $\ell_X \leftarrow \ell'$
11: **if** $\ell_X \leftarrow \infty$ **then**
12: **return** ∞
13: **return** $\max_X\{\ell_X\}$

例如，对于 3.4 而言，（Hap，Σfirst）处于无限的 ℓ-BCNF 层次结构当且仅当 ℓ≥1m 时处于正常形式。它与诸如 3NF、BCNF、4NF 等已知的范式正交。

已知结合硬度 CCs 结果和应用 FDs 提升到 ℓ-BCNF。在不增加（太多）计算复杂度的情况下提高表达能力。作为 BCNF 的一种推广，

3.1 局部高效可判定性

首先，我们证明对于给定的模式（R，Σ）以及给定的正整数 ℓ，判断（R，Σ）是否处于 ℓ-BCNF 可以在输入规模的三次方时间内完成。这是由于我们证明了只需检查 Σ 中的函数依赖即可验证（R，Σ）是否处于 ℓ-BCNF。

定理 3.5. 对于所有 （R， Σ） 和 ℓ∈N∞⩾1，（R， Σ） 在 ℓ-BCNF 中当且仅当对于所有 $X \rightarrow Y \in \Sigma$ 其中 $Y \not\subseteq X$，我们有 card (X) ⩽ℓ∈Σ_L+。对于给定的模式 （R， Σ） 和给定的 ℓ∈$N^{\infty}_{\geqslant 1}$，我们可以在 O（$|\Sigma|^2 \times ||\Sigma||$） 时间内判定 （R， Σ） 是否在 ℓ-BCNF 中。

3.2 计算最强局部范式

我们的新设定促使了这样一个问题的产生：计算给定模式（R，Σ）处于 ℓ-BCNF 时的最小正整数 ℓ。对于关系模式 R 上的 CC 和 FD 集合 Σ，我们用 ΣFD 表示 Σ 中的 FD 集合，用 Σcard 表示 Σ 中的 CC 集合。算法 1 计算出使（R，Σ）处于 ℓ-BCNF 的最小 ℓ。其步骤如下。

如果输入中没有函数依赖（第 1 行），则输出级别为 1（第 2 行）。否则，我们从最坏的情况开始，级别设为无穷大（第 3 行）。我们将 CC 卡（R）⩽1 加入给定的 CC 集合，因为 R 总是主键。对于每个非平凡的输入函数依赖，其左部为 X（第 5 行），我们确定根据输入 X 受限的最小正整数 ℓX（第 5 至 12 行）。初始时，ℓX 为无穷大（第 6 至 7 行），只要能找到某个 CC 卡（Y）⩽ℓ′∈Σcard（第 8 行），使得函数依赖 X →Y 成立（第 9 行），则当前的 ℓX 就可以用更小的 ℓ′ 替换（第 10 行）。实际上，在这种情况下，CC 卡（X）⩽ℓ′ 也由 Σ 推导得出（参见表 3 中的回拉规则 P）。一旦发现某个 X 的 ℓX 为无穷大（第 11 至 12 行），则立即终止并返回无穷大。否则，返回计算出的 ℓX 中的最大值（第 13 行）。

算法 1 计算出最强的 ℓ-BCNF，其中 ℓ 是具有该性质的最小正整数。本质上，对于输入中的每个函数依赖 X → Y，我们检查输入中的每个基数约束，看其是否蕴含某个更小的界限 ℓX 。因此，算法 1 的运行时间为 O（$|\Sigma|^2 \times ||\Sigma||$）。由此我们得到以下结果。

推论 3.6. 给定关系模式 R 上的 CC 和 FD 的集合 Σ，我们可以在 O（$|\Sigma|^2 \times ||\Sigma||$） 的时间内计算出最小的正整数 ℓ，使得（R， Σ） 处于 ℓ-BCNF 中。

对于 （Hap，Σ），有 ℓE = $_{1k}$，ℓV = $_{1m}$，ℓVT = $_1$，ℓET = $_1$ 以及 ℓCT = 1。由于最大值为 ℓ = ℓ2 = 1m，所以 ℓ2 是使（Hap，Σ）处于 ℓ-BCNF 形式的最优 ℓ。

3.3 全球范围内的可能难解性

虽然我们能够高效地在局部计算出最强范式，但在全局范围内这样做不太可能高效。给定（R，Σ，ℓ）以及 R 的子模式 S ⊂ R，我们不太可能找到一个能$_{在}$多项式时间内$_{判定}$（S，Σ[S]）$_{是否处于}$ ℓ-BCNF 的算法。

定理 3.7. 设 Σ 表示关系 R 上的一组 CC 和 FD，S ⊂ R，且 ℓ 为正整数。对于给定的 （R， S， Σ， ℓ），判定 （S， Σ[S]） 是否处于 ℓ-BCNF 是$_{共}$ NP 完全问题。

虽然判定给定的子模式是否处于 BCNF（1-BCNF）已经是 coNP 完全问题，但定理 3.7 鼓励我们从计算模式处于 ℓ-BCNF 时的最小 ℓ 这一新视角来审视模式规范化。

4 ℓ-BCNF 的有用属性

我们证明了在 ℓ-BCNF 中的模式刻画了具有更新和连接操作时有用属性的实例。特别是，ℓ 表示为实现数据一致性所需更新的最大值的数量，同时也表示给定冗余值可连接的数据值的数量。

4.1 ℓ冗余性

直观地说，文森特 [47] 将数据值的单次出现定义为冗余的，只要对该出现的每次更改都会导致关系违反某些给定的约束条件。我们的想法是选定某个正整数 ℓ，将数据值定义为 ℓ-冗余的，只要存在 ℓ 个不同的元组包含该值，并且对这 ℓ 个出现中的至少一个进行更新都会导致关系违反给定的约束条件。也就是说，这 ℓ 个出现的值已经由其他值以及关系满足约束条件的情况唯一确定了。

例如，在表 1a 中关系 r4 中的值“Kilo”是 999 次出现但不是 1000 次出现的冗余（ℓ 1 = 1000）：在 C 列中隐藏 1 到 999 次出现的值“Kilo”仍然允许我们确定每次隐藏的出现，因为 E → C 必须成立，并且至少还存在一个元组具有匹配的 E 值和 C 值“Kilo”，如下所示。

E	C	T
聚会，派对	？	t_1
..	..	..
·	·	·
聚会，派对	？	t_{99}
聚会，派对	千（重量单位）	

E → C

？ = 千（克）

E	C	T
聚会，派对	千（重量单位）	t_1
..	..	..
·	·	·
聚会，派对	千（重量单位）	t_{99}
聚会，派对	千（重量单位）	

然而，隐藏所有 1000 次出现意味着元组在 C 上可以具有任何值（只要它们在 C 上都匹配即可）。

我们通过规定不存在任何具有任何 ℓ 冗余的关联来定义一组语义范式。模式 （R， Σ） 处于 ℓ 冗余自由范式 （ℓ-RFNF） 当且仅当对于 R 上的任何关系 r 满足 Σ，不存在任何 σ∈Σ 以及任何属性 A ∈ R，使得存在数据值 v ∈r(A) 对于 σ 是 ℓ 冗余的。

示例 3.2 中的 （Hap， Σ） 属于 1m-RFNF（ℓ⨰ = 1m），但对于任何 ℓ < ℓ⨰ 都不属于 ℓ-RFNF。隐藏少于 ℓ⨰ 个“Mega”的出现次数，我们就能推断出它们的值。

更正式地说，对于数据值 v ∈ r(A) 而言，若存在 ℓ 个不同的元组 t1， . . . ， tℓ ∈ r 使得 t1(A) = . . . = tℓ(A) = v， 并且对于 t1， . . . ， tℓ 的每一个 ℓ 事务 {t_1'， . . . , tℓ′}，其中 t1， . . . ， tℓ 与 A 相关，关系 r′ : = (r − {t1, . . . , tℓ}) ∪ {t1′， . . . ， tℓ′} 违反了 σ，则称 v 对于 σ ∈ Σ 是 ℓ 冗余的。

一个 ℓ 事务会导致至少一个 ℓ 值的实际更新。更正式地说，对于 A 的 t1，...，tℓ 的 ℓ 事务是一个关于 R 的元组集 {t1′，...，tℓ′}，使得对于所有 i = 1，...，ℓ 以及所有 A′∈R - {A}，ti′（A′） = ti（A′），并且存在某个 j ∈ {1, ...，ℓ} 使得 t_j'(A) ， tj (A)。

4.2 更新效率低下

一个 ℓ 冗余数据值与 ℓ 更新低效在以下意义上是同义的。无论对任何 ℓ 个该 ℓ 冗余数据值的出现进行更新，都无法实现一致性，因为总会存在一些未被更新的该值的出现。对于上文右侧的表，无论我们如何更新 999 个 Kilo 的出现，只要更新了其中至少一个，就无法满足 E → C。然而，将 Kilo 的所有 1000 个出现一致地更新为新值则能满足 E → C。基于 CC 卡（E） ⩽ 1k，我们要求在任何合法关系中，为保持 E → C 的一致性，最多更新 1000 个元组。因此，可以说 （R， Σ） 处于 ℓ 更新低效范式 （UINF） 当且仅当 （R， Σ） 处于 ℓ 冗余函数范式（RFNF）。

如果关系模式 （R， Σ） 属于 ℓ-RFNF（ℓ-UINF），则 ℓ_{is} （R， Σ） 防止的数据冗余（更新低效）的程度为 ℓ。否则，（R， Σ） 允许该程度。如果不存在这样的 ℓ 使得 （R， Σ） 属于 ℓ-RFNF（ℓ-UINF），则对于 （R， Σ） 防止的程度不存在先验的上限。在这种情况下，设 ℓ : = ∞。

4.3 ℓ 连接效率

我们将连接效率定义为在非平凡函数依赖的左部具有匹配值的最大元组数量。这体现了由于函数依赖而带来的更新和连接效率，这是逻辑模式设计中的核心权衡。

形式上，如果关系模式 R 上的非平凡函数依赖 X→Y 在关系 r 上成立，则 r 是其在 XY 和 X（R−Y） 上投影的无损连接。即 r = πXY (r) ▷◁πX(R−Y)(r)。因此，对于关系 r 中的固定元组 t，其 X 值与所有在 X 上具有匹配值的元组 t1′， . . . ， tℓ′∈r 的唯一 Y 值以及 R−Y 值进行连接。这些元组的数量 ℓ 表示元组 t 在 X→Y 上的连接强度。关系 r 的连接强度是 r 中任何元组的连接强度的最大值。最后，（R， Σ） 的连接效率是满足 Σ 的 R 上任何关系的最大连接强度。模式 （R， Σ） 在 ℓ 连接效率范式（JENF）中，当且仅当 （R， Σ） 的连接效率不超过 ℓ。

例如，模式（ECT，{CT → E，E → C，card(E) ⩽ 1k}）的连接效率为 1k。下面的关系 r 将冗余的 C 值 Kilo 与 1k 个不同的值 t1，...，t1k 进行连接。由于 r 满足 E → C，因此我们有 r = r[EC] ▷◁ r[ET]。

E	C	T
Party	Kilo	t_1
⋮	⋮	⋮
Party	Kilo	t_{1k}

=

E	C
Party	Kilo

⋈

E	T
Party	t_1
⋮	⋮
Party	t_{1k}

如果 （R， Σ） 属于 ℓ-JENF，则 ℓ_{is} 表示 （R， Σ） 所允许的连接效率级别。否则，表示 （R， Σ） 所阻止的连接效率级别为 ℓ_{is}。如果不存在这样的 ℓ 使得 （R， Σ） 属于 ℓ-JENF，则对于 （R， Σ） 所允许的连接效率级别不存在先验的上限。在这种情况下，结果为 $\ell_{:=}$ ∞。

4.4 理由

结果表明，对于每个 ℓ∈N⩾∞1，ℓ-BCNF 与 ℓ-RFNF（ℓ-UINF）以及 ℓ-JENF 是一致的。因此，处于 ℓ-BCNF 的模式能够捕获到这样的实例：i) 没有任何 ℓ 冗余的数据值出现（ℓ 更新效率低下），并且 ii) 允许 ℓ 级别的连接效率。

定理 4.1. 对于所有 （R， Σ） 以及所有 ℓ∈ N∞

⩾1 时，以下为等价：（1） （R， Σ）处于 ℓ-RFNF（ℓ-UINF）状态，（2） （R，Σ）处于 ℓ-JENF 状态，以及（3） （R，Σ）处于 ℓ-BCNF 状态。

定理 4.1 的证明——尤其重要的是——为每一个不在 ℓ-BCNF 中的模式构造了一个实例，该实例具有某些 ℓ 冗余数据值出现（ℓ 更新低效）以及 ℓ + 1 级连接强度。这种构造在实践中可用于自动生成体现模式属性的关系。

示例 3.2 中的（Hap，Σ）处于 ℓ2-BCNF 但不在 ℓ2 - 1-BCNF 中。CC 卡（V） ⩽ℓ2 - 1 并非由 Σ 推导得出，但 V → C 是 Σ 中的一个函数依赖。基于此函数依赖，我们构建了一个包含 ℓ2 个元组 t1′，...，tℓ′ 的关系，这些元组在 V+Σ[FD] = VC 列上的值都匹配，而在其他所有列上具有 2 个不同的值。该关系可能由表 1a 中 r 的最后 ℓ2 个元组组成。它避免了 ℓ2 级别的数据冗余和更新低效，并允许 ℓ2 级别的连接高效。

5 模式设计算法

在定义了模式分解的更新低效性和连接高效性之后，我们提出了三种逻辑模式设计算法。我们还说明了我们的概念如何在逻辑设计阶段就为视图选择提供信息。

5.1 评估分解

我们可以评估模式分解 D 对更新和连接操作的质量。D 的连接效率仅由 D 保留的那些函数依赖决定。这些函数依赖可能已被转换为键，也可能未被转换。我们将支持连接操作的属性子集的集合定义为

$$JS_{\mathcal{D}}^{(R,\Sigma)} = \{S : X_k \mid \exists S \in \mathcal{D} \exists X \to Y \in \Sigma[S], \Sigma \models X \to S\} \cup \{S : X \mid \exists S \in \mathcal{D} \exists X \to Y \in \Sigma[S], \Sigma \not\models X \to S\}.$$

接下来，对于一个支持连接的属性集，其连接强度由适用于它的任何 CC 的最小上界给出。因此，对于 （R， Σ） 和 X ⊆ R，设 ℓX := min{ℓ | Σ|= card(X) ⩽ ℓ} 为 X 的最小数据冗余级别。现在我们定义 D 的连接效率级别为所有支持连接的属性集的最小数据冗余级别中的最大值，即

$$\ell_{\mathcal{D}}^{J} := \sup\{\ell_X \mid S : X \in JS_{\mathcal{D}}^{(R,\Sigma)}\} \cup \{1 \mid S : X_k \in JS_{\mathcal{D}}^{(R,\Sigma)}\}.$$

本质上，函数依赖 X → Y 转化为键 X 时贡献级别为 1，其他函数依赖贡献级别为 ℓX 。定义 ℓD J 为上确界
这意味着当Σ中不含任何函数依赖时其值为 1 。此外，D 的连接效率总水平是各水平之和:

$$\ell_{\mathcal{D}}^{J,\text{total}} = \sum_{S:X \in JS_{\mathcal{D}}^{(R,\Sigma)}} \ell_X + \sum_{S:X_k \in JS_{\mathcal{D}}^{(R,\Sigma)}} 1.$$

D 的更新效率由输入的所有函数依赖（FD）决定。特别是，任何丢失的函数依赖都需要在每次更新时通过连接 D 的元素来强制执行。将我们运行示例分解为 R2 和 R3 的 BCNF 分解会导致函数依赖 E → C 的丢失。如表 1b 中的实例 r2 和 r3 所示，r2 中任何 C 值（例如 Kilo）的更新都需要在 r2 中具有相同事件（例如 Party）的所有出现位置上保持一致，因此函数依赖 E → C 仍然成立（在 R2 和 R3 的连接上）。该示例说明了丢失的函数依赖如何导致模式连接时的数据冗余。这就是需要强制执行它们的原因。因此，我们将更新关键属性子集的集合定义为

$$UC_{\mathcal{D}}^{(R,\Sigma)} = JS_{\mathcal{D}}^{(R,\Sigma)} \cup \{X \subseteq R \mid \exists X \to Y \in (\Sigma - (\cup_{S \in \mathcal{D}} \Sigma[S])^{+})\}.$$

现在我们将 D 的更新效率低下程度定义为所有更新关键属性集的最小数据冗余度的最大值，即

$$\ell_{\mathcal{D}}^{U} := \sup\{\ell_X \mid S : X \in UC_{\mathcal{D}}^{(R,\Sigma)}\} \cup \{1 \mid S : X_k \in UC_{\mathcal{D}}^{(R,\Sigma)}\}.$$

或者，我们也可以求和来得出 D 的总体更新效率低下的程度，具体如下:

$$\ell_{\mathcal{D}}^{U,\text{total}} = \sum_{S:X \in UC_{\mathcal{D}}^{(R,\Sigma)}} \ell_X + \sum_{S:X_k \in UC_{\mathcal{D}}^{(R,\Sigma)}} 1.$$

仅使用函数依赖时，BCNF 分解或 3NF 合成可能会导致最优情况，即保持函数依赖的 BCNF 分解，其连接效率为 1，更新效率为 1。否则，某些函数依赖会丢失或不是键。因此，所有非最优情况都会导致更新效率和连接效率无界。而使用多集约束时，我们的框架可以在所有情况下测量更新效率和连接效率。

我们现在将提出不同的模式设计算法，并在后面通过实验展示它们的成果。

Algorithm 2 Opt(R, Σ)

Require: (R, Σ) with CC/FD set Σ over schema R
Ensure: Lossless, FD-preserving $\ell_{\mathcal{D}}$-BCNF decomposition $\mathcal{D}$ of (R, Σ) with minimum $\ell_{\mathcal{D}}$ $(= \ell_{\mathcal{D}}^{U} = \ell_{\mathcal{D}}^{J})$

1: Choose an atomic cover Σ_a of Σ [25];
2: **for all** $X \to A \in \Sigma_a$ **do**
3: $\quad \Sigma[X \to A] \leftarrow \emptyset$
4: $\quad$ **for all** $Y \to B \in \Sigma_a (YB \subseteq XA \wedge XA \not\subseteq Y^{+}_{\Sigma[FD]})$ **do**
5: $\quad\quad \ell_Y \leftarrow \min\{\ell \mid \Sigma \models card(Y) \leq \ell\}$
6: $\quad\quad \Sigma[X \to A] \leftarrow \Sigma[X \to A] \cup \{(Y \to B, \ell_Y)\}$
7: $\quad \ell_{X \to A} \leftarrow \sup\{\ell_Y \mid Y \to B \in \Sigma[X \to A]\}$
8: $\mathcal{D} \leftarrow \emptyset$;
9: **for all** $X \to A \in \Sigma_a$ in decreasing order of $\ell_{X \to A}$ **do**
10: $\quad$ **if** $\Sigma_a - \{X \to A\} \models X \to A$ **then**
11: $\quad\quad \Sigma_a \leftarrow \Sigma_a - \{X \to A\}$
12: $\quad$ **else**
13: $\quad\quad \mathcal{D} \leftarrow \mathcal{D} \cup \{(XA, \Sigma_a[XA])\}$
14: Remove all $(S, \Sigma_a[S]) \in \mathcal{D}$ if $\exists (S', \Sigma_a[S']) \in \mathcal{D} (S \subseteq S')$
15: **if** there is no $(R', \Sigma') \in \mathcal{D}$ where $R' \to R \in \Sigma^{+}_{\mathcal{U}}$ **then**
16: $\quad$ Choose a minimal key K for R with respect to Σ
17: $\quad \mathcal{D} \leftarrow \mathcal{D} \cup \{(K, \Sigma_a[K])\}$
18: **Return**$(\mathcal{D}, \ell_{\mathcal{D}})$

5.2 FD 保存

保留所有函数依赖可确保连接效率和更新效率的水平一致。否则，更新效率的水平可能会超过连接效率的水平（弊大于利）。因此，我们的目标是在所有保留函数依赖的分解中将更新效率的水平降至最低。为此，我们从唯一的原子覆盖（即所有右端为单属性的 L 约简函数依赖）开始，然后针对所有函数依赖 σ 计算与被 σ 包含的所有非主键函数依赖相关联的最大 ℓσ，接着按照 ℓσ 从大到小的顺序合成最终分解，除非该 σ 被剩余的函数依赖所蕴含。由于定理 3.7，算法 2 在最坏情况下呈指数级。

示例 3.2 中的 （Hap， Σ） 已经处于第三范式。由于不存在保持函数依赖的 BCNF 分解，经典范式化在此停止。将 （Hap， Σ） 作为算法 2 的输入，我们得到 Σa = Σ∪{CT →E}（第 1 行）。因此，ℓCT →V = 1m 和 ℓCT →E = 1k，对于其他 X →A ∈Σa ，ℓX→ A = 1（第 7 行）。然而，CT →V 可由 Σa -{CT →V } 推导得出（第 10 行）。这一步至关重要，因为 CT →V 实际上已被 CT →E 所取代，这也意味着由 CT →V 引起的 1m 级数据冗余已降低为由 CT →E 引起的 1k 级数据冗余。此外，R1 ⊆R4（第 13 行），所以算法 2 返回引言中的 1k-BCNF 模式 D1 = {(R2， Σ2), (R3， Σ3), (R4， Σ4)}。经典范式化既不会生成 D1 也不会生成 D2。即使生成了，D1、D2 和 （Hap， Σ） 也会被认为质量相同。

我们可以将算法 2 应用于 Σ 的部分输入。例如，排除那些可能造成高更新效率低下但实际更新不太可能违反的函数依赖，可能会提高连接效率。同样，也可以包含所有满足给定阈值 ℓX 的函数依赖 X → Y。

Algorithm 3 GREED(R, Σ)

Require: (R, Σ) with CC/FD set Σ over R, $\ell \in \mathbb{N}_{\geq 1}$
Ensure: Lossless $\ell^U_{\mathcal{D}}$-BCNF decomposition $\mathcal{D}$ of (R, Σ)
1: **if** (R, Σ) is in ℓ-BCNF **then**
2: $\mathcal{D} \leftarrow \{(R, \Sigma)\}$
3: **else**
4: $\ell_{\max} \leftarrow \max\{\ell_X \mid X \to Y \in \Sigma, Y \not\subseteq X, \Sigma \not\models X \to R\}$
5: Pick $X \to Y \in \Sigma\ (Y \not\subseteq X \wedge \Sigma \not\models X \to R \wedge \ell_X = \ell_{\max})$
6: $R_1 \leftarrow XY$; $R_2 \leftarrow X(R - XY)$
7: $\mathcal{D} \leftarrow$ GREED$(R_1, \Sigma[R_1]) \cup$ GREED$(R_2, \Sigma[R_2])$
8: **Return**$(\mathcal{D}, \ell^U_{\mathcal{D}}, \ell^J_{\mathcal{D}})$

5.3 贪婪分解

通常，FD 保持需要许多输出表，这与低连接效率相关。我们将经典的 BCNF 分解重新定位为一种使用更少表来减少 ℓUD 的贪心算法。选择驱动经典 BCNF 分解的非主键 FD 是任意的。有了 CC，我们选择数据冗余程度最大的 FD（例如，∞）。算法 3 展示了这一策略。为了提高连接效率，我们可以使用更高的 ℓ 作为输入。如果未指定，默认目标是 ℓ = 1。只要局部模式不在 ℓ-BCNF 中（第 1 行），算法就从一些剩余的最佳 FD 开始（第 4 - 5 行）。输出返回模式 D、它防止的 ℓUD 的更新效率级别以及 ℓD 的级别。 J

它允许的连接效率越高，输出表就越少，但代价是由于丢失了函数依赖（FDs），对这些层次的控制就会丧失。其中，当 ℓX = ∞ 时，FDs X → A 具有较高的优先级。若未给出任何连接条件（CCs），则属于经典的 BCNF 分解情况，此时 ℓUD = ∞ = ℓ D。 J

，除非 D 是保持 FD 的，且 ℓUD = 1 = ℓD J .

我们将算法 3 应用于示例 3.2 中的 （Hap， Σ）。由于 V → C ∈Σ 会导致 ℓmax = 1m（第 4 行），我们得到 Dд = {(R2 = VC， Σ2 = {V →C}), (R_3 = EVT， Σ_3 = {TV → E, ET →V })}（第 7 行）。 我们丢失了 E →C 和 CT →V，所以 ℓUD = 1k。我们创建断言检查

德

在连接 R2 ▷◁R3 上强制执行丢失的函数依赖 E →C 和 CT →V：

创建断言 LostFDEtoC 检查（不存在（从 R2、R3 中选择 R3.E 其中 R2.V = R3. V 按 R3.E 分组 使得计数（R2.C）> 1））； 创建断言 LostFDCTtoV 检查（不存在（从 R2、R3 中选择 R2.C、R3.T 其中 R2.V = R3.V 按 R2.C、R3.T 分组 使得计数（R2.V）> 1））；

Dд 在模式数量更少的情况下达到了与 D1 相同的更新效率低下水平。但这是以丢失需要断言检查的函数依赖以及较小的连接效率 ℓD 为代价的。

J ⊤ = 1.

当有更多信息可用时，我们的算法可以进行调整。例如，虽然 Dд 是由 FD V →C 驱动的，因为 card(V) ⩽ 1m，但可能不会基于 V 值更新 C 值，而是频繁基于 E 值更新 C 值。因此，我们可以优先考虑 FD E →C，从而得到 Dд′= {(R1 = EC， Σ1 = {E →C}), (R3， Σ3)}，丢失了 FDs V →C 和 CT →V 。

与保持 FD 的 BCNF 分解并非总能实现一样，对于给定的 ℓ，也并非总能实现 ℓ-BCNF。以一个经典示例 [3] 为例，考虑关系模式 R = {c(ity), s(treet), z(ip),A} ，其中

Algorithm 4 HYBRID(R, Σ)

Require: (R, Σ) with CC/FD set Σ over schema R
Ensure: Lossless $\ell^U_{\mathcal{D}}$-BCNF decomposition $\mathcal{D}$ of (R, Σ)
1: $\mathcal{D} \leftarrow$ OPT(R, Σ)
2: **for all** $S \in \mathcal{D}$ **do**
3: $S = XA$ for some $X \to A \in \Sigma_a$
4: **for all** $Y \to B \in \Sigma_a (YB \subseteq XA \wedge XA \not\subseteq Y^+_{\Sigma[FD]})$ **do**
5: **if** $\ell_Y > \ell_X$ **then**
6: $S_1 = YB$; $S_2 = Y(XA - B)$
7: $\mathcal{D} \leftarrow (\mathcal{D} - \{(S, \Sigma_a[S])\}) \cup \{(S_1, \Sigma_a[S_1]), (S_2, \Sigma_a[S_2])\}$
8: Remove any non-maximal schema from $\mathcal{D}$
9: **Return**$(\mathcal{D}, \ell^U_{\mathcal{D}}, \ell^J_{\mathcal{D}})$

Σ 包含 sc → z 和 z → c，且 CCs card(c,s) ⩽ ℓ + 1 以及 card(z) ⩽ ℓ + 1。若 {c， s， z} 不在输出 D 中，我们失去 sc → z，此时 ℓUD = ℓ + 1。若 {c， s， z} 在 D 中，我们在 {c, s， z} 上会有 z → c，因此 ℓUD = ℓ + 1。所以，没有任何输出能处于 ℓ-BCNF 中。

5.4 混合算法

我们将前两种策略结合到混合算法 4 中。首先应用算法 2（第 1 行），以获得所有保持 FD 的分解中最小的可能的 ℓD。接下来，我们检查输出模式 XA（第 2 至 3 行）中是否任何处于第三范式但不在 BCNF 中的模式（第 4 行），其上的非主键 FD Y →B 满足 ℓY > ℓX（第 5 行）。在这种情况下，我们通过以 FD Y →B 分解（第 6/7 行），用较低的更新效率级别 ℓX 来换取 X →A 的保持。

对于示例 3.2 中的输入（Hap，Σ），算法 4 将分解 D1 作为算法 2 的输出（第 1 行）。特别地，S = R4 = CET 处于第三范式但不在 BCNF，因为对于 Σa [S] = Σ4 = {CT → E, E → C}。也就是说，函数依赖 E → C ∈ Σa 对模式 S = R4 = CET 是关键的（第 3 行和第 4 行）。由于 ℓE = 1k > 1 = ℓCT（第 5 行），我们沿关键函数依赖 E → C 进行分解，用 R6 = EC 和 R7 = ET 替换 R4（第 6 行和第 7 行）。然而，R6 = R1 且 R7 ⊆ R3。因此，输出为 Dh = {(R1， Σ1), (R2， Σ2), (R3， Σ3)}，且 ℓUD = 1 = ℓDJ

断言检查 LostFDCTtoV 在连接 R2 ▷◁ R3 上强制执行 FD CT → V 。

表 4 总结了我们的算法为示例返回的模式以及 D2，包括它们的属性。

5.5 物化视图

在数据库运行时，会出现频繁的访问模式。添加物化视图有助于加速查询，但会占用额外的存储空间，并且需要花费时间来维护数据一致性。因此，视图的选择应基于量化连接支持和维护成本来进行。我们将给定视图 V（即一组属性）的（总）连接效率水平和更新效率低下程度定义为 ℓV（总）。

以及 ℓV U（总） ，分别。

以我们的混合模式 Dh 为例，常见的查询可能会询问公司在某个场地的工作时间。因此，我们可以引入以下物化视图 V。

CREATE MATERIALIZED VIEW $\mathcal{V}$ AS (
SELECT $R_2.V, R_2.C, R_3.T$ FROM R_2, R_3 WHERE $R_2.V = R_3.V$);

表 4：运行示例的模式设计

方法	模式	丢失的FDs	LU	J	物化视图
Opt	D1: R2 = CV 其中 V → C R_3 = ETV 其中 TV → E, ET → V R_4 = CET 其中 CT → E, E → C	无	一千	一千	VD1 = πVCT （R2 与 R3 并联）
N/A	D2: R1 = EC 且 E → C R_3 = ETV 且 TV → E, ET → V R_5 = CTV 且 CT → V, V → C	无	1 米	1 米	VD_2 = πECT (R_1 ▷◁ R_3)
贪婪	Dд：R_2 = CV 且 V → C R_3 = ETV 且 TV → E, ET → V	E → C CT → V	一千	1	Vdд = πECT (R_2 ▷◁ R_3))
混合动力的	Dh：R_1 = EC，其中 E → C；R_2 = CV，其中 V → C R_3 = 具有 TV 的 ETV → E, ET → V	CT → V	1	1	Vdh = πVCT (R_1 ▷◁ R_3))

表 5：运行示例的模式设计上的查询 q1 和 q2

q	D_1	D1 + VD1	D_2	D^2 + VD^2	德	Dд + vdд	Dh	Dh + VDh
$_1$ 问	πECT (R_4)	πECT (R_4) VD_1	πECT (R_1▷◁R_3))	VD_2	πECT (R_2▷◁R_3))	VDд	πECT (R_1 ▷◁ R_3) πECT (R_1 ▷◁ R_3)	
$_2$ 问	πCTV (R_2 ▷◁ R_3))		πCTV (R_5)	πCTV (R_5)	πCTV (R_2 ▷◁ R_3) πCTV (R_2 ▷◁ R_3)		πCTV (R_1 ▷◁ R_3))	VD_h

在这种情况下，断言检查 LostFDCTtoV 可以直接在视图上执行。

创建断言 LostFDCTtoV 检查（不存在（从 V 中选择 C, T）

按 C、T 分组，其中 V 的计数大于 1 的分组；

由于 V → C、CT → V 以及 card(V) ⩽ 1m，该视图分别具有 1m 级别的更新低效性和连接高效性。V
连接支持是有效的，但需要将基表 R2 上的 C 更新传播到视图 V 上多达 100 万次更新。因此，我们量化更新效率低下与连接效率之间的权衡也是物化视图的固有特性，应作为帮助选择视图的信息的一部分。表 4 还列出了我们运行示例的各种模式设计的一些视图。

6 实验

我们的实验分析了我们范式的特性以及这些特性如何转化为模式实例上更新和连接操作的性能。我们在 Visual C++ 中实现了我们的算法。实验是在一台配备 Intel Xeon W-2123 处理器（3.6 GHz）、256GB 内存、运行 Windows 10 操作系统的 PC 上进行的，使用的是 2017 年版的 SQL Server 社区版。

6.1 定性研究

首先，我们针对运行示例的模式设计，在合成实例上分析了两个更新操作和两个连接操作的性能。

我们关于（Hap，Σ）的第一个实例包含 1,001,000 个元组。其中 1k 个元组在 E 和 C 上的值匹配，1m 个元组在 V 和 C 上的值匹配。其他值在其所在列中都是唯一的。该实例与表 1a 中的实例 r 同构，其中 ℓ1 = 1k，ℓ2 = 1m。为了说明函数依赖对更新和连接的影响，它们在该实例中导致的冗余值的数量与模式上的数据冗余级别（ℓ1 和 ℓ2）一致。

对于实验，我们考虑了受我们的 CC 和 FD 影响的四种操作：

· 更新 u1 会替换与给定 E 值相关联的所有 C 值的出现。

· 更新 u2 会替换与给定 V 值相关联的所有 C 值的出现。

查询 q1 返回所有 CTE 组合，并且

查询 q2 返回所有 CTV 组合。

对表 4 中的每个模式设计，每个操作均运行 100 次，并报告平均值。

更新总是从基表传播到视图。由于在 Dд 上 E → C 丢失，u1 会在 R2 ▷◁ R3 上更新 C 值，并将其投影到 R2 上。表 5 展示了在表 4 中的每个模式下，查询 q1 和 q2 在没有和有物化视图时的样子。

在满足 3NF 范式的模式（Hap，Σ）的实例上，q1 和 q2 是简单的投影操作，不需要进行连接操作。对于 u1，由于非主键函数依赖 E→C，更新了 1k 次冗余的 C 值；对于 u2，由于非主键函数依赖 V→C，更新了 1m 次冗余的 C 值。

表 6 展示了在投影到表 4 的模式上的实例上执行操作所花费的时间（以秒为单位）。

具体更新和查询最好由不同的设计来支持。值得注意的是，对于函数依赖（FD），数据冗余的程度会按比例影响到受该函数依赖影响的更新和连接操作的性能。这同样适用于视图。

操作 u2 和 q1 在 D1 上使用 R4 和 R2 上的键 V 时执行速度快，但在 D2 上执行速度慢。相反，操作 u1 和 q2 在 D2 上使用 R5 和 R1 上的键 E 时执行速度快，但在 D1 上执行速度慢。对于每个操作，D1 和 D2 之间的性能差异与影响该操作的函数依赖所导致的数据冗余程度成正比。例如，函数依赖 V→C（其中 ℓV = 1m）导致在 D1 和 D2 上执行 u2 时存在显著差异（44.58 秒），以及在 D1 和 D2 上执行 q2 时存在差异（1.35 秒）。在

表 6：合成实验 1（单位：秒）

Op	哈普	D_1	D1 + VD1	D_2	$D^2 + VD^2$	德	Dд + vdд	Dh	Dh + VDh
1	0.93	0.91	0.91	0.59	1.03	1.62	1.81	0.59	0.59
2	45.75	0.23	45.33	44.81	44.81	0.20	0.20	0.21	83.82
1	0.011	0.009	0.009	0.17	0.012	1.32	0.009	0.14	0.14
2	3.39	4.11	3.47	2.76	2.76	4.09	4.09	4.32	3.44

表 7：合成实验 2（单位：毫秒）

Op	哈普	D_1	D1 + VD1	D_2	$D^2 + VD^2$	德	Dд + vdд	Dh	Dh + VDh
1	58.2	55.8	55.8	3.7	60.7	82.3	170.2	3.7	3.7
2	10	4.4	12.7	9.8	9.8	4.9	4.9	4.7	18.9
1 问	8.3	7.5	7.5	11.1	10.6	236	4	11.6	11.6
2	5.6	14.6	5.7	5.3	5.3	14.5	14.5	14.1	7.4

相比之下，当 ℓV = 1k 时，FD E →C 导致在 D1 和 D2 上执行 u1（0.32 秒）以及在 D1 和 D2 上执行 q1（0.161 秒）之间的差异要小得多。

相对于 ℓVD1 = 1 米，VD1 对 q2 的加速作用和对 u2 的减速作用较大。相对于 ℓVD2 = 1 千米，VD2 对 q1 的加速作用和对 u1 的减速作用较小。

不出所料，Dд 在 u2 上表现良好，但在其他操作上则不然。VDд 加快了 q1 的速度，但对 u1 的性能影响较小，不过 u1 本来性能就差。

对于 u1 和 u2，Dh 表现良好，对于 q1 表现也不错。VDh 虽然加快了 q2 的速度，但显著减慢了 u2 的速度。

1100 表元组，7 显示 ℓ1 = 我们的 1k，结果 ℓ2′= 对于 100 的 < 第二 1m。合成实验实例说明，与第一个实例相比，相同的操作组合由相同的设计最佳支持。由于实际基数（100）远小于允许的基数（1m），因此处理具有此基数的操作的时间要短得多（以毫秒为单位）。

逻辑设计的选择取决于人们认为其对未来操作工作负载的支持程度。我们的框架通过设计或物化视图中函数依赖所表现出的数据冗余程度来量化这种支持，这衡量了它们对特定更新和连接操作性能的影响。相比之下，传统的逻辑设计可能无法提出某些设计，并且无法量化对工作负载的支持。例如，示例 3.2 中的模式（Hap，Θ）处于第三范式，不存在无损且保持函数依赖的分解到 BCNF。因此，仅基于函数依赖的传统设计会在此停止。相反，通过将 CC 包括在分析中，我们的方法将示例 3.2 中的模式（Hap，Σ）转换为表 4 中列出的各种模式设计，并从更新效率低下和连接效率方面量化了这些设计的成果。

6.2 定量研究

其次，我们分析了基于从表 8 中的现实世界数据集挖掘出的函数依赖（FD）和连接条件（CC）所推导出的模式设计的更新和连接性能。它展示了发现的函数依赖数量（#FD）、空标记出现的总次数（#⊥）以及行数和列数（#R、#C）。

表 8：实验中使用的数据集

数据集	#函数依赖（FDs）	#⊥	#R	#C
中国	918	418580	262920	18
nc选民	568	3079822	1024000	19

这些数据集对从数据中发现函数依赖 X → A 的基准算法进行了测试，并根据数据中满足 card(X) ⩽ ℓX 的最小下界 ℓX 以降序排列这些算法[41, 42, 48]。我们将同一列中出现的不同空值视为匹配的域值（⊥ = ⊥）。否则，我们的结果非常相似（⊥,⊥）。所发现的 CC 和 FD 表示数据中的模式，但不一定代表域中的模式。我们的算法输入仅是更相关的 FD，即排名在前 20% 的那些。添加更多的 FD 几乎不会造成任何差异，这表明具有更高数据冗余度的 FD 决定了模式设计。

6.2.1 更新和连接效率水平分析。我们将 Algorithms 2（最优）、3（贪婪）和 4（混合）应用于 ncvoter 和 china 的模式。

表 9 展示了分解 D 和原始输入的 ℓU；ℓJ 水平，它们的总水平 ℓU，total；ℓJ，total 、输出大小 |D| 以及运行时间。

对于“Opt”和“Hybrid”，更新效率低下的程度显著降低。“Greed”由于丢失了部分函数依赖，仅实现了较小程度的降低。由于每个函数依赖要么成为键，要么丢失，“Greed”的连接效率水平等于 1。在函数依赖保持的情况下，“Opt”始终能达到最优的更新效率水平 ℓU，且由于所有函数依赖均得以保留，该水平始终等于 ℓJ 。而“Hybrid”通过丢失部分函数依赖以及降低 ℓJ，实现了更低的更新效率水平 ℓU 。
贪婪算法通过减少模式的数量来实现不同的输出规模的最优值：通过添加模式来保留所有函数依赖，而混合算法则通过添加更多模式来实现进一步的减少。由于输入为原子覆盖，所有算法都运行迅速[25]。

6.2.2 物理性能。我们在原始数据集以及算法 2 和算法 3 的输出结果上运行了更新和连接查询，输入集包含了所有满足给定目标级别 ℓ 的函数依赖 X → Y。图 3 和图 5 展示了算法 2 的结果，以及

表 9：在实际模式上的算法成就

措施	请给出需要翻译的词汇。	最优的	贪婪	混合动力的
我爱你	四千零九十七	15	512	7
你好	四千零九十七	15	1	7
总长度 U	五千零四十三	87	三千零三十五	79
总长度 J	五千零四十三	87	6	73
|D|	1	65	6	66
时间（毫秒）	-	$<_1$	$<_1$	$<_1$

（a）针对 ncvoter 中排名前 20% 的活跃用户的措施

措施 ℓU	请给出需要翻译的词汇。	最优的	贪婪	混合动力的
总电荷量 ℓJ,	四千八百八十九	56	四千八百八十一	55
总电荷量 ℓJ,	四千八百八十九	56	1	50
总电荷量 ℓJ,	六万二千四百九十二	327	五万四千二百四十七	257
总电荷量 ℓJ, 总	六万二千四百九十二	327	4	165
|D|	1	145	4	157
时间（毫秒）	-	$<_1$	$<_1$	$<_1$

（b）针对中国前 20%富裕家庭的措施

图 3：使用 Opt 后的连接性能

图 4：使用 Greed 后的连接性能

图 4 和图 6 展示的是算法 3 的结果。x 轴始终表示目标级别 ℓ、输出表的数量 n 以及输入函数依赖（FD）的数量。图 3 和图 4 的 y 轴表示连接查询的运行时间（以秒为单位），而图 5 和图 6 的 y 轴表示更新操作的运行时间（以秒为单位）。

连接查询如下：

· 从原始模式 Hap 中选择所有列，以及

· 从 R_1，...，Rn 中选择$_{所有}$列，执行 R1 自然连接 . . . 自然连接 Rn 。

随着目标级别 ℓ 的降低，连接时间增加。对于 Opt，由于需要许多输出表来保持函数依赖关系，其连接性能变差。这在 Greed 中得到了补偿，但代价是丢失函数依赖关系时更新性能不佳。由输入完整性约束单独决定的级别 ℓ 会转化为输出更新和连接的不同物理性能程度。

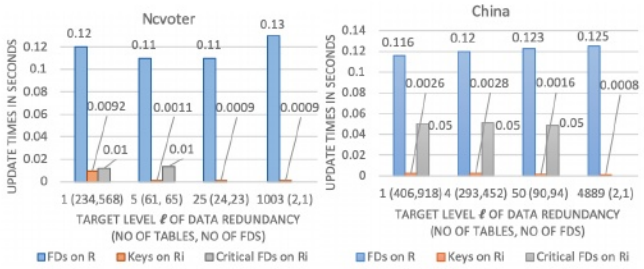


图 5：使用 Opt 后的更新性能

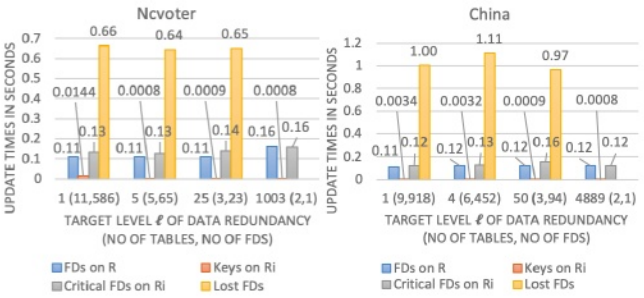


图 6：使用 Greed 后的更新性能

这是我们新框架的目的所在，即提供这些选项。

更新涉及对某些冗余值的所有出现位置进行更改（回想一下第 4 节中的更新场景）。在将函数依赖转换为键之后，每个值最多出现一次，因此最多只需要对一个出现位置进行更新。

并且平均 6 次更新表明，对于约 ℓ 级别的键，每次更新时间降低约两个数量级，比原始数据集上的输入 FD 的更新时间低。对于保留的非键 FD（称为关键 FD），其收益约为一个数量级（图 5 和 6 中的灰色条形图，当所有 FD 都成为键时没有数据点）。丢失的 FD 需要比原始数据集更多的更新时间，这是由于必要的连接操作。如果没有丢失的 FD，只需强制执行键（在图 6 中针对最大目标级别）。图 5 将 FD 转换为键（对于 Opt），这提高了更新效率，但代价是引入了更多的表。一个主要信息是，模式设计主要由高排名的 FD 驱动，但包含低排名的 FD 意味着在合成过程中需要保留更多的 FD 以降低其更新时间（通过转换为键或关键 FD 使用 Opt），或者在分解过程中会丢失更多的 FD（Greed）。

6.2.3 评论。逻辑模式设计由业务规则驱动，这些规则将实例限制在对底层领域有意义的范围内。我们已经看到，CC 的上限如何影响模式设计并量化对查询和更新操作的支持。我们的实验表明，上限越小，不同设计之间的性能差异就越小。在逻辑设计期间，模式的选择因此取决于上限，因为这些上限代表了当前的需求。当需求随时间变化时，物理数据库调优（例如反规范化）可以提升性能。然而，这仅限于

一旦数据库投入运行，能够观察到实际实例并且工作负载的稳定模式已经显现，就有可能进行调整。如果约束条件发生变化，以至于另一种设计更为合适，那么这可能就证明有必要迁移到新的模式。最终，逻辑设计会根据应用程序的需求来决定模式的选择，而不论这些需求是否已经发生变化。

对设计进行排序有助于寻找相关的模式，就如同对网站进行排序有助于寻找相关的网站一样。盲目选择具有最小 ℓ 的设计类似于使用“我运气好”按钮。这同样适用于物化视图的选择。我们的实验表明，应仔细检查哪些操作会受到哪些函数依赖的影响。由函数依赖引起的数据冗余程度量化了它们对操作的影响程度。特别是，如果某些函数依赖引起的数据冗余程度事先是无界的（$\ell = \infty$），我们的框架会提醒分析人员需求分析可能仍不完整。

7 相关工作

模式设计已在诸如 SQL [26]、嵌套关系 [39]、数据仓库 [30]、面向对象 [24]、语义 [9]、时态性 [23]、网络 [1]、不确定性 [35] 以及图 [16] 等数据模型中得到研究。与经典设计类似，我们的结果可扩展到更丰富的数据模型和更高的范式，例如 4NF [14]、投影连接范式 [15] 或包含依赖范式 [33]。

与仅使用函数依赖（FD）的 BCNF [4, 5] 和 3NF [6] 不同，CC 用于衡量实现数据一致性所需的工作量。我们的算法返回的设计能够量化最坏情况下的更新低效性和最佳情况下的连接效率。我们的概念还量化了逻辑设计期间物化视图的增量维护和连接支持。相比之下，数值依赖（ND）在水平分解为满足 FD 的块之后应用经典范式化 [19]。ND 不是有限公理化的 [18]。

模式设计并不期望为所有实例优化布局。相反，在部署逻辑模式之后，一旦可靠的数据检索模式出现，物理调优就会发挥作用。这包括主内存数据库中的虚拟反规范化 [37]、迁移到 NoSQL [49] 以及数据仓库设计 [30]。已经制定了从反规范化模式恢复 3NF 的指南 [43]，以及对规范化模式进行反规范化的方法 [7]。信息论为雪花模式中的事实表 [32] 和 3NF [28] 提供了理论依据。更新效率低下和连接效率影响了选择物化视图这一难题 [11]。

科伊奇和米利切夫通过最大化读取收益，同时将写入成本保持在阈值以下来实现去规范化[27]。他们的技术基于特定的更新和查询来调整逻辑模式（例如，处于第三范式）。因此，他们的方法适用于数据库已运行且有成熟的工作负载模型可用的情况。这种额外的输入使得能够为输入的工作负载推导出优化的模式[27]。[27]中的工作未使用CC。我们的方法是针对逻辑模式规范化，此时频繁更新和查询操作的工作负载模型尚未可用。我们使用CC来探索具有不同更新效率和连接效率属性的各种范式模式。因此，设计团队可以根据这些属性评估不同的设计。我们仅需要算法和 CC 2 的输出属性集（Hap，Σa）作为输入。由于我们的逻辑模式是基于3NF提出的（Hap，Σa），它可能为应用[27]中的工作提供与经典规范化不同的起点。例如，一旦频繁更新和查询出现，我们可以应用[27]。

针对 NoSQL 数据库的模式设计和演进的近期方法 [45] 是由查询工作负载驱动的，因此 NoSQL 模式是非规范化的。我们尚未发现有关用于 CC 和 FD 的逻辑 NoSQL 模式设计的工作。

基数约束是概念建模中最重要的约束类型之一[40]。“基数约束对应于关系上非常常见的语义规则，在概念层面上对其进行形式化定义能显著提高数据描述的完整性”[31]。令人惊讶的是，我们应用基数约束进行逻辑模式设计的方法此前未曾被观察到。基数约束在 Chen 的 ER 经典论文[9]中首次提出，并在诸如语义[34]、Web[17]、空间和时间[13]以及不确定模型[44]等数据模型中得到了研究。它们是包括 UML、EER、ORM、XSD（如 maxOccurs）或 OWL（如 owl:maxCardinality）[20]在内的主要数据和知识建模语言的一部分。它们还被用于数据清理[10]、数据库测试[8]、查询回答[12]和逆向工程[46]。

脏数据对依赖关系的违反促使了诸如近似键和近似函数依赖等宽松概念的提出[22, 29]。从（脏）数据中挖掘时，近似概念可能会提高识别有意义规则的召回率，但会降低其精确度。CC $\text{card}(X) \leqslant \ell$ 可以被视为允许最多 ℓ 个重复项的近似键。因此，对于较小的 ℓ，我们的工作可以看作是将经典模式设计扩展到近似键，为脏数据提供了一定的鲁棒性。

8 结论与未来工作

我们提出了首个仅基于完整性约束来衡量更新低效性和连接高效性的逻辑模式设计框架。这得益于新的“级别 ℓ 数据冗余”概念，该概念由模式设计时 CC 的上限决定。我们提出的无限个 ℓ 有界基数范式族，刻画了无级别 ℓ 数据冗余和更新低效性，并允许级别 ℓ 连接高效性的实例。我们开发了模式设计算法，并通过实验展示了它们如何在输出设计规模上进行权衡，从而降低更新低效性和连接高效性的级别。我们还通过实验展示了这些级别如何量化模式设计和物化视图对于特定更新和连接操作在实例上的性能适用性。我们的框架利用关于 CC 的领域知识来推进逻辑模式设计。

未来的工作将处理更多的约束条件和数据模型。我们预计，CC 与连接依赖之间的相互作用将对更高范式的开发构成挑战。

致谢

作者感谢匿名审稿人提出的宝贵建议。第一作者感谢 Joachim Biskup、Bernhard Thalheim 和 Jef Wijsen 在 19031 次达格斯图尔研讨会上的富有见地的讨论。

参考文献

[1] Marcelo Arenas. 2006. Normalization theory for XML. SIGMOD Record 35, 4 (2006), 57–64.

[2] William Ward Armstrong. 1974. Dependency Structures of Data Base Relation-ships.. In IFIP congress, Vol. 74. Geneva, Switzerland, 580–583.

[3] 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.

[4] Catriel Beeri, Philip A Bernstein, and Nathan Goodman. 1978. A sophisticate’s introduction to database normalization theory. In VLDB. 113–124.

[5] Philip A. Bernstein and Nathan Goodman. 1980. What does Boyce-Codd Normal Form Do?. In VLDB. 245–259.

[6] Joachim Biskup, Umeshwar Dayal, and Philip A. Bernstein. 1979. Synthesizing Independent Database Schemas. In SIGMOD. 143–151.

[7] Douglas B. Bock and John F. Schrage. 2002. Denormalization guidelines for base and transaction tables. ACM SIGCSE Bull. 34, 4 (2002), 129–133.

[8] Nicolas Bruno, Surajit Chaudhuri, and Dilys Thomas. 2006. Generating Queries with Cardinality Constraints for DBMS Testing. IEEE Trans. Knowl. Data Eng. 18, 12 2006), 1721–1725.

[9] Peter Chen. 1976. The entity-relationship model-toward a unified view of data. ACM Trans. Database Syst. 1, 1 (1976), 9–36.

[10] Wenguang Chen, Wenfei Fan, and Shuai Ma. 2009. Incorporating cardinality constraints and synonym rules into conditional functional dependencies. Inf. Process. Lett. 109, 14 (2009), 783–789.

[11] Rada Chirkova and Jun Yang. 2012. Materialized Views. Found. Trends Databases 4, 4 (2012), 295–405.

[12] Graham Cormode, Divesh Srivastava, Entong Shen, and Ting Yu. 2012. Aggregate Query Answering on Possibilistic Data with Cardinality Constraints. In ICDE. 258–269.

[13] Faiz Currim and Sudha Ram. 2008. Conceptually modeling windows and bounds for space and time in database constraints. Commun. ACM 51, 11 (2008), 125–129.

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

[15] Ronald Fagin. 1979. Normal Forms and Relational Database Operators. In SIGMOD. 153–160.

[16] Wenfei Fan, Yinghui Wu, and Jingbo Xu. 2016. Functional Dependencies for Graphs. In SIGMOD. 1843–1857.

[17] Flavio Ferrarotti, Sven Hartmann, and Sebastian Link. 2013. Efficiency frontiers of XML cardinality constraints. Data Knowl. Eng. 87 (2013), 297–319.

[18] John Grant and Jack Minker. 1985. Inferences for Numerical Dependencies. Theor. Comput. Sci. 41 (1985), 271–287.

[19] John Grant and Jack Minker. 1985. Normalization and Axiomatization for Nu-merical Dependencies. Inf. Control. 65, 1 (1985), 1–17.

[20] Neil Hall, Henning Köhler, Sebastian Link, Henri Prade, and Xiaofang Zhou. 2015. Cardinality constraints on qualitatively uncertain data. Data Knowl. Eng. 99 (2015), 126–150.

[21] Sven Hartmann. 2003. Reasoning about participation constraints and Chen’s constraints. In ADC. 105–113.

[22] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. 1999. TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependen-cies. Comput. J. 42, 2 (1999), 100–111.

[23] 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.

[24] Vitaliy L. Khizder and Grant E. Weddell. 2003. Reasoning about Uniqueness Constraints in Object Relational Databases. IEEE Trans. Knowl. Data Eng. 15, 5 (2003), 1295–1306.

[25] Henning Köhler. 2006. Finding Faithful Boyce-Codd Normal Form Decomposi-tions. In AAIM. 102–113.

[26] Henning Köhler and Sebastian Link. 2016. SQL Schema Design: Foundations, Normal Forms, and Normalization. In SIGMOD. 267–279.

[27] Nemanja Kojic and Dragan Milicev. 2020. Equilibrium of Redundancy in Rela-tional Model for Optimized Data Retrieval. IEEE Trans. Knowl. Data Eng. 32, 9 (2020), 1707–1721.

[28] Solmaz Kolahi and Leonid Libkin. 2010. An information-theoretic analysis of worst-case redundancy in database design. ACM Trans. Database Syst. 35, 1 (2010), 5:1–5 32.

[29] Sebastian Kruse and Felix Naumann. 2018. Efficient Discovery of Approximate Dependencies. Proc. VLDB Endow. 11, 7 (2018), 759–772.

[30] Jens Lechtenbörger and Gottfried Vossen. 2003. Multidimensional normal forms for warehouse design. Inf. Syst. 28, 5 (2003), 415–434.

[31] Maurizio Lenzerini and Gaetano Santucci. 1983. Cardinality Constraints in the Entity-Relationship Model. In ER. 529–549.

[32] Mark Levene and George Loizou. 2003. Why is the snowflake schema a good data warehouse design? Inf. Syst. 28, 3 (2003), 225–240.

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

[34] Stephen W. Liddle, David W. Embley, and Scott N. Woodfield. 1993. Cardinality Constraints in Semantic Data Models. Data Knowl. Eng. 11, 3 (1993), 235–270.

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

[36] S. Link and Z. Wei. 2021. Logical Schema Design that Quantify Update Inefficiency and Join Efficiency. Report CDMTCS-552. The University of Auckland. http: //www.cs.auckland.ac.nz/research/groups/CDMTCS/

[37] Zezhou Liu and Stratos Idreos. 2016. Main Memory Adaptive Denormalization. In SIGMOD. 2253–2254.

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

[39] Wai Yin Mok, Yiu-Kai Ng, and David W. Embley. 1996. A Normal Form for Precisely Characterizing Redundancy in Nested Relations. ACM Trans. Database Syst. 21, 1 (1996), 77–106.

[40] Antoni Olivé. 2007. Cardinality Constraints. In Conceptual Modeling of Informa-tion Systems. Springer, 83–102.

[41] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. 2015. Func-tional Dependency Discovery: An Experimental Evaluation of Seven Algorithms. PVLDB 8, 10 (2015), 1082–1093.

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

[43] Jean-Marc Petit, Farouk Toumani, Jean-François Boulicaut, and Jacques Kouloumdjian. 1996. Towards the Reverse Engineering of Denormalized Re-lational Databases. In ICDE. 218–227.

[44] Tania Roblot, Miika Hannula, and Sebastian Link. 2018. Probabilistic Cardinality Constraints. VLDB J. 27, 6 (2018), 771–795.

[45] Stefanie Scherzinger and Sebastian Sidortschuck. 2020. An Empirical Study on the Design and Evolution of NoSQL Database Schemas. In ER. 441–455.

[46] Christian Soutou. 1998. Relational Database Reverse Engineering: Algorithms to Extract Cardinality Constraints. Data Knowl. Eng. 28, 2 (1998), 161–207.

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

[48] Ziheng Wei and Sebastian Link. 2019. Discovery and Ranking of Functional Dependencies. In ICDE. 1526–1537.

[49] Jaejun Yoo, Ki-Hoon Lee, and Young-Ho Jeon. 2018. Migration from RDBMS to NoSQL Using Column-level Denormalization and Atomic Aggregates. J. Inf. Sci. Eng. 34, 1 (2018), 243–259.