arXiv:2301.04846v1 [cs.LO] 12 Jan 2023

代数模型管理：综述

Patrick Schultz[1], David I. Spivak[1], 和 Ryan Wisnesky[2]

[1] 马萨诸塞州理工学院[2] Conexus AI

摘要。我们综述模型管理领域，并介绍一种基于代数规范的新型模型管理方法。

1 引言

本文综述模型管理领域，并介绍一种借鉴代数规范领域技术的新型模型管理方法，希望借此在这两个领域之间建立一种中间语言。我们所说的“模型管理”，指的是 Bernstein 和 Melnik 所说的“元数据密集型”数据库管理 [4], ，我们将在第2节对其进行定义。我们所说的“新型代数模型管理方法”，是指我们使用代数（等式）理论 [19] [20]来描述数据库模式和实例的特定方法。

我们在研究将范畴论 [3] 应用于数据集成 [5]的过程中，首次注意到模型管理与代数规范之间的联系。这些研究成果见于 [19] 和 [20], ，本文不提出实质性的新结果。我们假设读者具备范畴论 [3]、代数规范 [17], 和 SQL 的基本知识。

概要。 第2节介绍模型管理的传统方法，第3节介绍我们的代数方法。在第3节中，我们还介绍开源的 CQL（范畴查询语言）工具，该工具可从 http://categoricaldata.net 下载，并以软件形式实现了我们的方法。最后，在第4 节中，我们将我们的方法与传统方法进行比较。

2 模型管理

引用 Melnik [16]:

> 信息系统工程面临的许多棘手问题都涉及复杂元数据制品（或模型）的操作，例如数据库模式、接口规范或对象图，以及模型之间的映射。解决元数据操作问题的应用程序十分复杂，难以构建。通用模型管理旨在提供一种数据库基础设施，将一组高级代数算子（如匹配、合并和复合）整体应用于模型和映射，而非其各个组成部分，从而减少开发此类应用程序所需的编程工作量。

上文将“模型”一词定义为模式等元数据制品，这与“模型”一词作为满足某种理论的结构的定义相冲突。本文用“模型管理”这一短语指代上文所述的领域，并用“模型”一词表示满足某种理论的结构。

如今，模型管理已成为信息管理的一个大型子领域，其研究文献包含数百篇已发表的文章 [4]。该领域的研究文献普遍认为 [4] 模型管理至少涉及下一节所述的问题。

2.1 模式映射

给定两个数据库模式 S 和 T，模式映射 问题 [7] 是构造一个“映射” F： $S \to T$ 用于捕获用户在 S 与 T之间指定的某种关系。不同的模型管理系统采用不同的模式概念，包括 SQL、XML 和 RDF [4]。研究文献中最常见的映射形式体系是“嵌入依赖”（EDs） [5]：一阶逻辑某个片段中的公式，具有实用的计算性质。

本节的示例将使用 SQL 模式和 EDs。考虑以下 SQL 模式 S，它由两个通过外键连接的表组成：

CREATE TABLE N2(ID INT 主键, age INT)

CREATE TABLE N1(ID INT 主键, name 字符串, salary INT, f INT 外键
REFERENCES N2(ID))

以及以下 SQL 模式 T，由一个表组成：

创建表 N（ID 整数 主键，年龄 整数，姓名 字符串，
薪水 整数）。

这两个 SQL 模式以图形方式展示在图 1中。

模式映射示例 $F : S \to T$，表示目标表 N 是沿着列 f 对源表 N1 和 N2 进行连接所得：

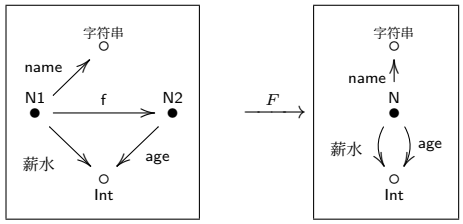
$$\forall id_1, id_2, a, n, s.\ \mathsf{N1}(id_1, n, s, id_2) \wedge \mathsf{N2}(id_2, a) \to \mathsf{N}(id_1, a, n, s).$$

图 1展示了两个满足上述存在依赖的实例。一般而言，多个存在依赖可以在两个 SQL 模式之间建立映射。

2.2 查询生成

给定一个模式映射 F： $S \to T$，查询生成问题 [4] 是构造一个查询，将 S 上的数据库转换为 T 上的数据库，并满足 F。通常研究的查询语言包括 SQL、XQuery，以及各种基于理解和 λ演算的语言 [4]。

用于实现第2.1 节示例映射的 SQL 查询如下：



N1
标识符 姓名 薪水 f
1 Alice $100 1
2 Bob $250 2
3 Sue $300 3

N2
标识符 年龄
1 20
2 20
3 30

$\leftarrow \llbracket \Delta_F \rrbracket$
$\rightarrow \llbracket \Pi_F \rrbracket \llbracket \Sigma_F \rrbracket$

N
标识符 姓名 薪水 年龄
1 Alice $100 20
2 Bob $250 20
3 Sue $300 30

图 1. 示例数据迁移：含外键（参见第 2.1、3.2节）

INSERT INTO N
SELECT N1.ID, N1.年龄, N2.姓名, N1.sal
FROM N1, N2 WHERE N1.f = N2.ID

严格来说，上述 SQL 代码中的 `INSERT` 部分并非“查询”，而是“更新”。在实践中，由查询生成任务生成的代码通常会存储查询结果。运行上述 SQL 的示例如图 1所示，方向为从左到右。一般而言，任意数量（包括零条）的 SQL 查询都可能实现一组存在依赖 [5]。存在依赖也可以通过一种称为“追赶算法”的算法直接执行 [5]。

2.3 映射求逆

给定模式映射 $F\colon S \to T$，映射求逆问题 [10] 是构造一个模式映射 $F^{-1}\colon T \to S$，使其相对于查询生成撤销 F（即从 F 和 F^{-1} 生成的查询应互为逆）。

对第 2.1 节中的模式映射求逆的自然候选存在依赖表示 N 投影到 N1 和 N2：

$$\forall id_1, a, n, s.\ \mathsf{N}(id_1, a, n, s) \to \exists id_2.\ \mathsf{N1}(id, n, s, id_2) \wedge \mathsf{N2}(id_2, a)$$

该存在依赖的一种可能 SQL 实现是：

插入到 N1
选择 ID、姓名、薪水、ID
来自 N

插入到 N2
SELECT 标识符, 年龄
FROM N

然而，通过取 $\emptyset$ =N1 ≠N2可以看出，上述存在依赖并不是第2.1节中存在依赖的逆，因为 [10], 在相关文献中，存在依赖或存在依赖集合可逆的情况很少见；“准逆”等较弱的逆概念 [10]则较为常见。运行上述 SQL 的一个示例显示在图1的从右到左方向中。

2.4 映射复合

给定模式映射 F： $S \to T$ 和 G： $T \to U$，映射复合问题 [8] 是构造一个模式映射 $G \circ F$： $S \to U$ 使其相对于查询生成问题等价（即，运行由 $G \circ F$ 生成的查询，应与先在 F生成的查询结果上运行由 G生成的查询具有相同的效果）。

第2.1 节中的存在依赖与第 2.3 节中的存在依赖复合后为

$$\forall id_1, id_2, n, s, a.\ \mathsf{N1}(id_1, n, s, id_2) \wedge \mathsf{N2}(id_2, a) \to \exists x.\ \mathsf{N1}'(id, n, s, x) \wedge \mathsf{N2}'(x, a)$$

其中 N1'、N2' 是源表 N1、N2的目标“副本”。这个复合后的存在依赖不是恒等映射，由此可见，第2.3节中的存在依赖并不能反转第 2.1节中的存在依赖。对于存在依赖，复合映射可能不存在 [8]，但存在依赖的某些限制和扩展在复合运算下是封闭的 [8]。

2.5 模式匹配

给定两个数据库模式 S 和 T，模式匹配问题 [5] 是自动查找 S 与 T 之间的“对应关系”，并根据这些对应关系自动推断模式映射 $S \to T$。一般而言，无法完全自动推断完整映射；匹配问题的重点是减少构建模式映射所需的人工工作量，例如，提出可由用户补全的部分映射。模式匹配技术多种多样，从利用字符串相似度比较列名，到使用机器学习算法；概述请参见 [5]。在第 2.1节的示例中，容易自动找到的两个对应关系是 (N1，N) 和 (N2，N)；Clio 等工具 [14] 可以根据这两个对应关系构造第 2.1 节中的存在依赖。

2.6 进一步参考文献

本文将重点讨论前几节所述的问题，但模型管理文献中还研究了许多其他问题 [4],其中许多问题与代数规范相关。例如，模式/实例合并问题 [4],常见于数据集成场景 [2],可以形式化为适当的模式/实例范畴中的推出 [20], 而此类推出与模型论概念相关，例如 模型粘合 [15]。

许多软件产品都用于解决模型管理问题 [4]，包括*ETL*（提取、转换、加载）工具 [5]，这类工具从不同数据库中提取数据，应用用户指定的转换，然后将结果加载到目标系统（如数据仓库）中；查询中介器 [5]，这类中介器通过组合针对不同源数据库的查询，来回答有关“虚拟”集成数据库的查询；以及可视化模式映射工具 [14] 这类工具允许用户通过在视觉上用连线连接相关的模式元素来创建模式映射，如图2所示。

至少有两种尝试，旨在为模型管理操作提供一种“元语义”。在 [16] 中，Melnik 通过将模式映射定义为 $S \to T$ 上的实例与 S 上的实例之间的任意二元关系，为上述部分操作提供了一种“基于状态”的元语义 T；前文所述的基于 ED 的语义就是这种元语义的一种具体化。在 [2] 和 [13] 中，作者通过将模式映射定义为适当的模式范畴中的态射，为上述部分操作提供了一种“机构理论”元语义 $S \to T$；CQL 的语义就是这种元语义的一种具体化。

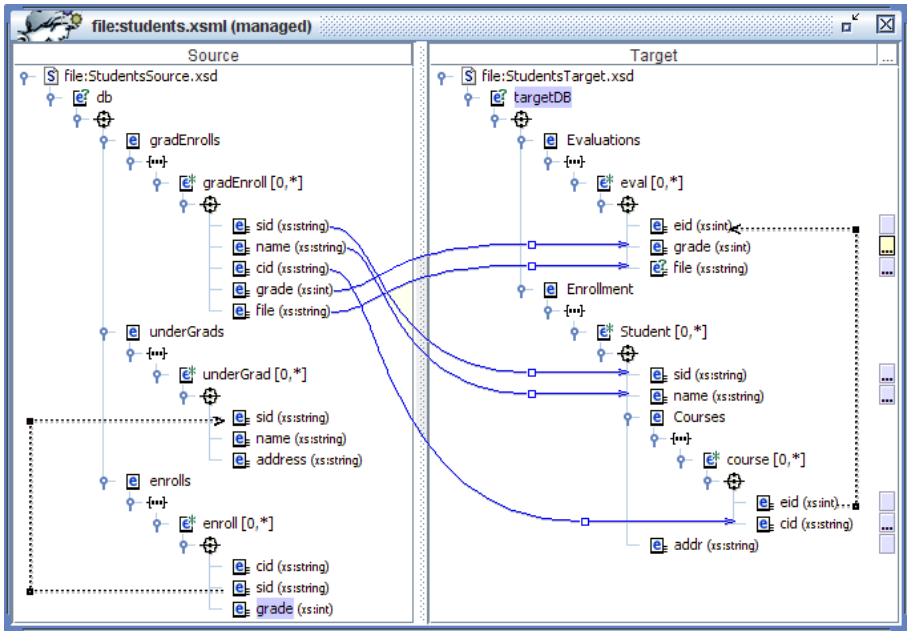


图 2. Clio 中的模式映射 [14]

3 代数模型管理

我们的模型管理方法基于我们在 [19]和 [20]中所述的数据库、数据迁移和数据集成的代数方法。这些工作以及本文都扩展了一种特定的范畴论数据模型，该模型起源于 20 世纪 90 年代末 [11]，后来在 [21] 和 [23]中得到扩展，并以 CQL 实现（http://categoricaldata.net）。

下一节将介绍我们用于数据库模式和实例的形式体系，并引入 CQL。后续各节将使用我们的形式体系，实现第2 节中的模型管理操作。本节中，我们将“代数理论”简称为“理论”。

3.1 代数数据库

在我们的形式体系中， [20],数据库模式和实例被定义为某种特定类型的理论，我们将在后续各节中加以说明。为便于阐述，我们有时会将形式体系中定义的模式和实例与其 CQL 等价形式混为一谈。

类型侧首先，我们固定一个理论 Ty，称为我们形式体系的类型侧。Ty 中的排序称为类型，而 Ty 中的函数则是可出现在模式和实例中的函数。

CQL 允许将任意理论用作类型侧。但我们发现，在实践中，CQL 用户几乎总是希望将某种现有编程语言的理论（例如 Java）用作类型侧。将 CQL“绑定”到现有语言的能力对于模型管理尤为重要，因为输入数据可能只能通过某种接口（例如 Java API）访问。为此，CQL 允许通过指定每种排序 s 对应的 Java 类 C_s，以及一个 Java 函数 String $\to C_s$ 来定义类型侧；该函数会告知 CQL 如何将其在 CQL 程序中遇到的字符串解释为 C_s 的对象。

图3展示了一个关于整数和字符串的 CQL 类型侧示例。该类型侧定义了一个包含两种排序和无限多个常量（所有 Java 字符串和整数）且没有方程的理论。关于 `Int` 的 Java 代码规定：当 CQL 程序中遇到字符串 x，且需要一个排序为 `Int` 的项时，应将 Java 的 `parseInt` 函数应用于 x，以得到所需的 `Int`。关键字 `literal`在 CQL 中多处使用，表示字面量（用户定义常量）的定义。

模式 A 模式 建立在类型侧 Ty 之上，是一个扩展 Ty 的理论，包含新的排序（称为实体）、从实体到类型的新一元函数（称为属性）、从实体到实体的新一元函数（称为外键）以及新方程（称为数据完整性约束），其形式为 $\forall v : s.\ t = t'$，其中 s 是一个实体，而 t、t' 是同一类型的项，每个项都包含一个自由变量 v。前一句中的限制（例如，不允许存在从类型到实体的函数）是将我们的形式体系用于模型管理所必需的 [19] [20]。

图 4展示了与图 1相对应的 CQL 模式。这些模式不包含任何方程，并且都位于图 *3* 中定义的类型侧 Ty 上。

```
类型侧 Ty  =字面量 {
    Java_类型
        字符串 = "java.lang.String"
        整数  ="java.lang.Integer"
    Java_常量
        字符串 = "return input[0]"
        整数  ="return java.lang.Integer.parseInt(input[0])"
}
```

图 3. CQL 类型侧 Ty

```
模式S  = 字面量 : Ty {
    实体
        N1
        N2
    外键_
        f : N1 ->N2
    属性
        姓名 : N1 -> 字符串
        薪水 : N1 ->整数
        年龄 : N2 -> 整数
}
```

```
模式 T  = 字面量 : Ty {
    实体
        N
    属性
        姓名 : N ->字符串
        薪水 : N -> 整数
        年龄 : N ->整数
}
```

图 4. CQL 模式 S 和 T 在类型侧 Ty

```
实例 I  =字面量 : S {
    生成元
        1 2 3 : N1
    方程
        姓名(1)  =Alice    薪水(1)  = 100    年龄(f(1))  = 20
        姓名(2)  =Bob      薪水(2)  = 250    年龄(f(2))  = 20
        姓名(3)  =Sue      薪水(3)  = 300    年龄(f(3))  = 30
}
```

图5. CQL 模式 S上的实例 I

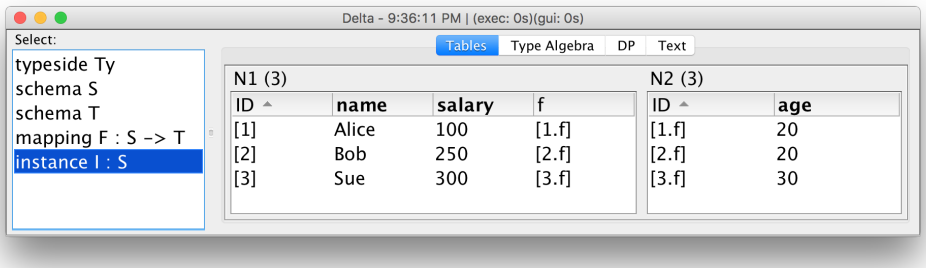


图6. CQL 实例 I的初始代数

实例 一个 实例 I 在模式 S 上是一个扩展了 S 的理论，其中包含称为生成元 的新零元函数（常量）符号，以及非量化方程。模式 S 上的 CQL 实例（图4）如图 5所示。

实例 I 的预期含义，记作 $[\![I]\!]$，是 I 的项模型 （即初始代数），其中对每个排序 s，都包含一个载体集，该集合由排序为 s 的闭项组成，并以 I 中的可证明性为准取商。实例态射 $I \to J$ 是代数 $[\![I]\!] \to [\![J]\!]$ 的同态（自然变换）。

图 6 展示了 I （来自图 5 ）在 CQL 工具中的含义。CQL 工具将项模型直观地显示为一组表，每个实体对应一张表 e，每张表都有一个 ID 列，对应于 e 的载体集。图 6 中的表与图 1中的左侧表同构。

在接下来的各节中，我们将使用前文对模式和实例的定义，实现第 2 节中的模型管理操作。

3.2 模式映射

给定模式 S、T，模式映射问题（第 2.1 节）是构造一个“映射” $F\colon S \to T$，以刻画 S 与 T 之间的某种关系。

设 S 和 T 是定义在同一类型侧 Ty 上的 CQL 模式。CQL 模式映射 $F\colon S \to T$ 定义为从S 到 T 的一个“导出签名态射” [18]，它在 Ty 上是恒等映射。也就是说，$F\colon S \to T$ 将每个实体$e \in S$映射到一个实体 $F(e) \in T$，并将每个属性/外键 $f\colon s \to s'$ 映射到一个项 $F(f)$，该项的类型为 $F(s')$，并且含有一个类型为 $F(s)$的自由变量，同时还要保持等式：如果 $S \vdash t = t'$，则 $T \vdash F(t) = F(t')$。我们发现，实践中出现的许多映射无法用普通签名态射表示，因此需要采用更一般的“导出”签名态射概念。

第 2.1 节中的模式映射是一个存在依赖（一阶逻辑某个片段中的公式），它在实例之间诱导出单一的二元满足关系；而 CQL 模式映射是导出签名态射，并在实例之间诱导出三种关系，我们将在下一节中加以说明。

一个 CQL 模式映射示例 $F\colon S \to T$ 如图 7所示，其中 CQL 模式 S 和 T 定义于图 4。该映射也以图形方式显示在图 1中。

3.3 查询生成

给定一个映射 $F\colon S \to T$，查询生成问题（第 2.2节）是使用 F 构造一个查询，将 S 上的数据库转换为 T上的数据库。

在我们的形式体系中，模式 S 上的数据库实例及其态射构成一个范畴，记作 S–Inst；模式映射 $F\colon S \to T$ 通过代换诱导出一个函子 $\Sigma_F\colon S\text{–Inst} \to T\text{–Inst}$。该函子 Σ_F 有一个右伴随 $\Delta_F\colon T\text{–Inst} \to S\text{–Inst}$；当用机构理论术语描述我们的形式体系时，它对应于“模型约化函子” [2]。函子 Δ_F 有一个右伴随 $\Pi_F\colon S\text{–Inst} \to T\text{–Inst}$。关于 Δ_F 始终存在左、右伴随函子的证明，参见 [19] 。作为伴随函子，Δ_F、Π_F 保持极限，而 Δ_F、Σ_F 保持余极限，因此可推出许多有用性质；例如，$\Sigma_F(I + J) \cong \Sigma_F(I) + \Sigma_F(J)$，以及 $\Pi_F(I \times J) \cong \Pi_F(I) \times \Pi_F(J)$。

```
映射 F = 字面量 : S ->T {
    实体
        N1 ->N
        N2 ->N
    外键_键
        f ->lambda x:N. x
    属性
        姓名 -> lambda x:N. name(x)
        薪水 -> lambda x:N. salary(x)
        年龄 ->lambda x:N. age(x)
}
```

图 7. CQL 模式映射 $F\colon S \to T$

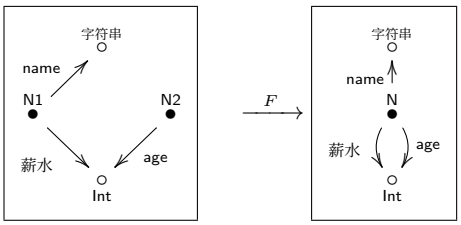
请注意，与2.1节不同，该节中一个模式映射只对应一个查询（存在依赖）；而在我们的代数方法中，共有三个查询，分别对应 Δ_F、 Σ_F和 Π_F。关于 Δ_F、Σ_F和 Π_F 何时可以用 SQL 表达，反之亦然，参见 [23]。

虽然可以给出定义 $\Delta_F, \Sigma_F, \Pi_F$ [19]的显式公式，但我们改为在图1和图8中给出示例。请注意，在这些示例中，我们没有展示第3.1节中定义的实例（理论），而是展示项模型。因此，在这些示例中，我们用指称括号 $[\![]\!]$ 括起 $\Delta_F, \Sigma_F, \Pi_F$。此外，由于伴随函子 Δ, Σ, Π 仅在唯一同构意义下定义，因此我们任意为标识符取名，并在这些示例中使用。图1和图8展示了一个 CQL 模式映射 F，它将两个不同的源实体 N1 和 N2 映射到目标实体 N。$[\![\Delta_F]\!]$ 函子投影的方向与 F 相反：它将列从 N 的单个表投影到 N1 和 N2 的两个独立表中，这类似于 SQL 中的 FROM N AS N1 和 FROM N AS N2。当 N1 到 N2 存在外键时，$[\![\Delta_F]\!]$ 函子会填充该外键，使得可以通过连接 N1 和 N2 恢复 N。当 N1 和 N2 之间不存在外键时，$[\![\Pi_F]\!]$ 函子会取 N1 和 N2 的笛卡尔积；若存在外键，则沿该外键连接 N1 和 N2。$[\![\Sigma_F]\!]$ 函子对 N1 和 N2 作不交并；由于 N1 和 N2 并不并兼容（列不同），$[\![\Sigma_F]\!]$ 会创建空值。当 N1 和 N2 之间存在外键时，$[\![\Sigma_F]\!]$ 会合并由该外键关联的元组，从而得到一个连接。正如这些示例所示，Δ_F 可以视为投影，Π_F 可以视为先取积再筛选（这可能产生连接），而 Σ_F 可以视为先作不交并（不要求并兼容），再合并（也可能产生连接）。

3.4 映射复合

给定模式映射 $F\colon\ S \to T$ 和 $G\colon\ T \to U$，映射复合问题（第 2.4 节）是构造一个模式映射 $G \circ F\colon\ S \to U$，使其在查询生成方面与给定映射等价。

从某种意义上说，对于我们的形式体系，映射复合问题微不足道 [19]：$\Delta_{F \circ G} \cong \Delta_G \circ \Delta_F$，$\Pi_{F \circ G} \cong \Pi_F \circ \Pi_G$，以及 $\Sigma_{F \circ G} \cong \Sigma_F \circ \Sigma_G$。但是



N1

标识符	姓名	薪水
1	Alice	$100
2	Bob	$250
3	Sue	$300

N2

标识符	年龄
4	20
5	20
6	30

$\xleftarrow{[\![\Delta_F]\!]}$

N

标识符	姓名	薪水	年龄
1	Alice	$100	20
2	Bob	$250	20
3	Sue	$300	30

N1

标识符	姓名	薪水
1	Alice	$100
2	Bob	$250
3	Sue	$300

N2

标识符	年龄
4	20
5	20
6	30

$\xrightarrow{[\![\Sigma_F]\!]}$

N

ID	name	薪水	age
1	Alice	$100	年龄(1)
2	Bob	$250	年龄(2)
3	Sue	$300	年龄(3)
4	姓名(4)	薪水(4)	20
5	姓名(5)	薪水(5)	20
6	姓名(6)	薪水(6)	30

N1

标识符	姓名	薪水
1	Alice	$100
2	Bob	$250
3	Sue	$300

N2

标识符	年龄
4	20
5	20
6	30

$\xrightarrow{[\![\Pi_F]\!]}$

N

标识符	姓名	薪水	年龄
1	Alice	$100	20
2	Bob	$250	20
3	Sue	$300	20
4	Alice	$100	20
5	Bob	$250	20
6	Sue	$300	20
7	Alice	$100	30
8	Bob	$250	30
9	Sue	$300	30

图 8. 示例数据迁移（参见第 3.2）

这种解决方案并不完全令人满意，因为在实践中，可能需要混合使用 Δ、 Σ和 Π函子来完成任何特定任务（类似地，在 SQL 中，可能需要混合使用连接和并集来完成任何特定任务）。以下结果在 [19] 和 [23]中得到了证明：

– 每个复合 $\Sigma_F \circ \Delta_G$都同构于某个 $\Delta_{F'} \circ \Sigma_{G'}$，其中 F'、 G'满足相应条件。如果将 Σ_F 替换为 Π_F.，这一结论仍然成立。– 形如 (F, G) 的对表示 $\Sigma_F \circ \Delta_G$，且在复合运算下封闭。如果将 Σ_F 替换为 Π_F，这一结论仍然成立。这类对可以用直观的“select-from-where”语法指定，详见 [19] 和 [20]。– 形如 (F, G, H) 的三元组表示 $\Sigma_F \circ \Pi_G \circ \Delta_H$；若 F 是离散反纤维化 [3], ，则它们在复合运算下封闭，而这恰好就是 [5]所要求的“并集兼容性”： Σ_F 对列相匹配的表进行并集；图1不是离散反纤维化。

3.5 映射求逆

给定一个模式映射 $F:\ S \to T$，映射求逆问题（第 2.3节）是构造一个映射 $F^{-1}: T \to S$，以某种方式“撤销” F。

我们的形式体系具有很强的求逆性质，但并不存在严格意义上的逆。当存在 $F^{-1}: T \to S$，使得 $F \circ F^{-1} = id$ 和 $F^{-1} \circ F = id$ 时，则有 $\Delta_F \circ \Delta_{F^{-1}} \cong id$、$\Sigma_F \circ \Sigma_{F^{-1}} \cong id$ 和 $\Pi_F \circ \Pi_{F^{-1}} \cong id$。一般而言， F 不一定有逆；但当 S 和 T 具有有限初始代数/项模型（这事先是不可判定的，且意味着 S 和 T 的可判定性）时，可以通过考察所有可能的函子 $T \to S$，在 F^{-1} 存在时将其构造出来。当 F 具有右伴随 $G: T \to S$（这是一个比存在逆更弱的条件）时，存在典范态射 $\Sigma_F \to \Delta_G$ 和 $\Delta_F \to \Pi_G$。

在实践中，即使逆不存在，数据的“往返” [5]仍然是可取的。例如，投影会遗忘信息，因此通常无法求逆，但我们可能希望记住投影后的数据源自何处。在我们的形式体系中， Σ、Δ和Π 之间的伴随提供了往返机制。例如，对于每个 $F: S \to T$ 以及每个 S实例 I ，都存在一个典范态射$I \to \Delta_F(\Sigma_F(I))$，即 $\Sigma_F \dashv \Delta_F$ 伴随的单位，它描述了 I中的每个标识符如何被 Σ_F映射（ Π_F也同理）。对偶地，对于每个 T实例 J ，都存在一个典范态射 $\Sigma_F(\Delta_F(J)) \to J$，即$\Sigma_F \dashv \Delta_F$ 伴随的余单位，它描述了 $\Delta_F(J)$中的标识符源自何处（ Π_F也同理）。对于每个态射$h: \Sigma_F(I) \to J$，都可以利用单位和余单位得到一个伴随态射 h': $I \to \Delta_F(J)$，反之亦然（Π_F也同理）。将伴随性与现有的宽松逆概念（例如准逆 [9]）联系起来，是未来研究的一个重要方向。

3.6 模式匹配

给定数据库模式 S 和 T，模式匹配问题（第2.5节）旨在自动向用户推荐模式映射 $S \to T$。

本节定义 CQL 使用的两种模式匹配技术。这些技术通过名称（即字符串）来比较实体、外键和属性（“符号”），因此依赖于名称（可能由用户提供），且这些名称的字符串相似度能够反映其语义相似度。令σ：字符串，字符串 $\to [0,1]$ 为任意字符串相似度函数 [5] 其中，值为 1 表示“匹配良好”，值为 0 表示“匹配较差”。

– 第一种技术尝试推断模式映射 $F\colon S \to T$。对于每个实体 $s \in S$，我们定义 $F(s) := t$ 其中 $t \in T$ 是使$\sigma(s,t)$ 最大化的实体。对于每个符号 $f\colon s \to s' \in S$，我们接着考虑符号集合 X，其中的符号 $F(s) \to F(s')$。如果 X 非空，我们选择使 $\sigma(f,g)$ 最大化的符号 $g \in X$，并设 $F(f) := g$。如果 X 为空，但存在一条从 $F(s)$ 到 $F(s')$ 的最短路径 p，我们设 $F(f) := p$。如果不存在最短路径，则匹配失败。如此构造的 F 仅是一个候选模式映射：CQL 必须验证 F 是否保留了 S 中的可证明相等。

– 第二种技术尝试推断一个模式 A 以及模式映射$F\colon A \to S$ 和 $G\colon A \to T$。这样的映射跨度可以解释为形式为 $\Sigma_F \circ \Delta_G$ 或 $\Pi_F \circ \Delta_G$的查询。设 c 为用户提供的某个字符串相似度阈值。A 的实体是满足 $\sigma(s,t) > c$的 S-实体与T-实体对 (s,t)。A的符号对 $(s,t) \to (s',t')$ 是满足$\sigma(f,g) > c$的 S-符号与 T-符号对 $(f\colon s \to s', g\colon t \to t')$。映射 F和 G 是投影。

4 结论

将我们的模型管理代数方法与源自关系数据库理论的其他方法进行比较时，[1] 需要注意的是，我们的数据库是“演绎数据库” [1]。也就是说，我们以“内涵”的方式将数据库定义为方程集，而非表集。因此，在协调我们的定义与关系型定义时必须谨慎。例如，我们的实例可能会出现“不一致”：某个实例可以证明 $1 = 2$ 两个不同的常量符号 1 和 2。此类情况通常是错误，但并非总是如此 [12], ；CQL 工具使用基于“保守理论扩张” [12]的标准技术来检查此类情况。此外，与关系数据库理论中的惯例不同，[7]我们的模式并未定义一组由该模式上的所有实例共享的常量（即“论域”）。因此，与数据库理论相比，[6] 我们的研究方法在精神上更接近传统逻辑 [1]。

我们的模型管理代数方法与第 2 节所述的基于 ED 的方法之间有许多联系。存在依赖比我们纯粹的等式数据完整性约束更具表达力，而且可以通过一种简单的方式将其纳入我们的形式体系，具体描述见 [22]（不过在 [22], 中，存在依赖称为“提升问题”）。在基于存在依赖的方法中，“追赶算法” [5] 操作的语义与我们的 Σ 操作类似；关于追赶算法与 Σ 的形式化比较即将发表。

致谢。作者感谢 Lucian Popa、Eswaran Subrahmanian 和 Peter Gates。本研究获得了美国国家标准与技术研究院小企业创新研究资助 70NANB 16H178、美国空军科学研究办公室资助 FA9550–14–1–0031 以及美国国家航空航天局资助 NNL14AA05C。

本文发表于 WADT 2016：代数开发技术近期趋势，第 56–69 页。

参考文献

1. Abiteboul, S., Hull, R., Vianu, V.：《数据库基础》。Addison-Wesley-Longman（1995） 2. Alagic, S., Bernstein, P.：通用模式管理的模型论。DBPL（2001） 3. Barr, M., Wells, C.：《计算机科学范畴论》。Prentice Hall International（1995） 4. Bernstein, P.A., Melnik, S.：模型管理 2.0：操作更丰富的映射。ICMD（2007） 5. Doan, H., Halevy, A., Ives, Z.：《数据集成原理》。Morgan Kaufmann（2012） 6. Enderton, H.B.：《逻辑数学导论》。Academic Press（2001） 7. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.：数据交换：语义与查询回答。《理论计算机科学》（2005） 8. Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.：组合模式映射：二阶依赖来解围。TODS（2005） 9. Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.：模式映射的拟逆。TODS（2008） 10. Fagin, R.：模式映射求逆。TODS（2007） 11. Fleming, M., Gunther, R., Rosebrugh, R.：范畴数据库。《符号计算杂志》35(2)（2003） 12. Ghilardi, S., Lutz, C., Wolter, F.：我是否损害了本体？《知识表示与推理原理》（2006） 13. Goguen, J.：机构中的信息集成（未发表）。http://cseweb.ucsd.edu/~goguen/pps/ifi04.pdf（2004） 14. Haas, L.M., Hern´andez, M.A., Ho, H., Popa, L., Roth, M.：Clio 的成长：从研究原型到工业工具。ICMD（2005） 15. Hodges, W.：《简明模型论》。剑桥大学出版社（1997） 16. Melnik, S.：《通用模型管理：概念与算法》（计算机科学讲义）。施普林格出版社（2004） 17. Mitchell, J.C.：《程序设计语言基础》。麻省理工学院出版社（1996） 18. Mossakowski, T., Krumnack, U., Maibaum, T.：什么是导出签名态射？RTADT（2014） 19. Schultz, P., Spivak, D.I., Vasilakopoulou, C., Wisnesky, R.：代数数据库。《范畴论及其应用》（2017） 20. Schultz, P., Wisnesky, R.：代数数据集成（未发表）。http://arxiv.org/abs/1503.03571（2016） 21. Spivak, D.I.：函子数据迁移。《信息与计算》（2012） 22. Spivak, D.I.：通过提升问题处理数据库查询与约束。《计算机科学中的数学结构》（2014） 23. Spivak, D.I., Wisnesky, R.：函子数据迁移的关系基础。DBPL（2015）