Lecture 6: 关系数据库设计

Database

Author: Forliage

Email: masterforliage@gmail.com

Date: June 11, 2025

College: 计算机科学与技术学院

Abstract

本讲笔记系统地介绍了关系数据库设计的核心理论与实践方法，旨在帮助读者通过规范化原则与依赖分析，构建无冗余且能保持约束的关系模式。内容共分八大部分：
1.第一范式 (1NF)：定义原子属性，分析处理复合属性和多值属性的策略及其缺点。
2.关系设计中的陷阱：剖析糟糕设计带来的冗余与异常，阐明分解的目标与理论基础。
3.函数依赖 (FD)：给出 FD 的形式定义、与键的关系，介绍闭包计算、Armstrong 公理与规范覆盖方法。
4.关系分解：说明无损连接和依赖保持的必要性，以及良好分解的判定条件。
5.Boyce–Codd 范式 (BCNF)：定义 BCNF，讲解测试与分解算法，并讨论与依赖保持之间的权衡。
6.第三范式 (3NF)：提出在保证依赖保持下的最小放宽，给出 3NF 定义、检测与分解算法，并通过示例说明其应用。
7.多值依赖 (MVD)：定义 MVD 的形式语义与推导规则，探讨典型案例中的冗余，并展示通过分解消除多值依赖的方法。
8.第四范式 (4NF)：在 MVD 基础上提出 4NF 定义，说明分解要求与算法，并简要展望更高范式（如 5NF、DKNF）。
通过本讲笔记的学习，读者将全面掌握关系数据库设计中的范式理论与算法，实现模式分解时的无损性与依赖保持，为高质量数据库模式设计与优化奠定坚实基础。
（该Abstract由ChatGPT-o4-mini-high生成）

Contents

1 第一范式

1.1 第一范式概念

定义：如果一个relation中所有属性的domain都是atomic的，则该关系属于第一范式(1NF);

什么是atomic：如果一个域中的元素被认为是不可再分的单元，这个域是atomic。

非原子域的例子：

- 复合属性，例如name的属性是name set
- 多值属性，例子一个人的多个pjone
- 复杂数据类型，面向对象中复杂的对象

关系数据库要求：所有关系必须复合1NF，即不能直接存储非原子值。

1.2 如何处理非原子值

处理复合属性：

- 使用多个属性来表示复合属性中的各组成部分
- 例如name → first name, last-name

处理多值属性：

- 多字段：person(pname,phone1,phone2,phone3,...) 缺点：字段数目不定，扩展困难。
- 单独表：(推荐)phone(pname, phone)每个phone值对应一行，结构灵活、可扩展。
- 单字段：person(pname, phones)将多个值拼接到一个字段中，不推荐，违反原子性，难以查询。

最合理做法是单独建表来处理多值属性(符合1NF，保持数据库设计规范)

1.3 非原子策略的缺点

如果没有正确处理非原子值，会导致以下问题：

- 存储复杂：数据结构不一致，难以维护。
- 鼓励冗余存储：多字段/拼接字段容易产生冗余。
- 查询复杂：SQL 查询变复杂，降低性能。

因此，设计中应尽量避免非原子值，确保符合 1NF。

1.4 原子性是使用方式决定的

关键观点：原子性不仅是数据的本质，还取决于使用方式。

示例：

- Strings通常被视为不可分
- 但如果学号CS0012/EE1127被拆解前两位判断部门→实际上这个字段就不是原子字段了
- 这种处理方式会导致在应用程序中编码信息，而不是在数据库中建模，这是不好的设计习惯

原因：数据库设计应清晰建模每个语义单元，不要把编码逻辑混入数据字段，避免隐式依赖，提升可维护性。

2 关系数据库设计中的陷阱

2.1 概述

设计目标：找到一组“好”的关系模式（relation schemas），以保证数据的存储和操作的质量。

坏设计（Bad design）会导致：

- 冗余存储
- 插入/删除/更新异常
- 信息丢失或难以表达

示例:关系模式 Lending-schema：(branch-name, branch-city, assets, customer-name, loan-number, amount) 该模式中混合了分支信息 + 客户信息 + 贷款信息，设计上存在冗余和异常风险。

2.2 分解

主精炼技术（refinement technique）：将一个大表（如 ABCD）拆分成多个子表：(AB), (BCD) 或 (ACD), (ABD) 等。

分解的要求：

- 所有原始属性必须出现在分解后的表中，即$R = R_1 \cup R_2$
- 无损连接分解（Lossless-join decomposition）：任意r(在R上的关系),应有：$r = \pi_{R_1}(r) \bowtie \pi_{R_2}(r)$

2.3 设计目标和理论基础

目标：

- 判断一个关系 R 是否是“好”的形式（Good form）：无冗余（No redundant）。
- 若 R 不是 good form，需将其分解为：$\{R_1, R_2, ..., R_n\}$ 使得:每个子关系都在 good form。整体分解是无损连接分解（Lossless-join decomposition）。

理论依据：

- 函数依赖:决定属性之间的约束关系。
- 多值依赖:处理属性取多值时的依赖关系。

3 函数依赖

3.1 什么是函数依赖

函数依赖(FD)是一种完整性约束，用于描述关系中属性之间的值的依赖关系。

数学上的类比是$x \to f(x)$，意味着一个属性组α的值可以唯一决定另一个属性组β的值，记作:$\alpha \to \beta$

定义：若对于关系R中任意两个元组t_1, t_2。若$t_1[\alpha] = t_2[\alpha]$，则$t_1[\beta] = t_2[\beta]$，则称$\alpha \to \beta$在$R$上成立。

例子：贷款信息表:F={loan-number→amount,loan-number→branch-name,(customer-name,loan-number)→amount,(customer-name,loan-number)→branch-name} 不希望出现loan-number→cutomer-name

3.2 函数依赖与键的关系

键是函数依赖的一种特殊形式

超键：K是R的超键↔ $K \to R$

候选键：$K \to R$，且K没有真子集α使得$\alpha \to R$

函数依赖比键更一般，可以表达键无法表达的约束。

3.3 函数依赖的作用

1.验证关系是否满足约束：给定关系r和函数依赖集F，若r满足F，则称r满足F

2.规范化设计模式：

- 判断模式的规范化程度，帮助消除冗余、提高设计质量。

- 可以推导出哪些属性组是否应该归属于同一张表。

3.4 平凡与非平凡依赖

平凡依赖：$\alpha \to \beta$若$\beta \subseteq \alpha$

非平凡依赖：$\alpha \to \beta$且$\beta \nsubseteq \alpha$

3.5 函数依赖集的闭包F^+

闭包是指：由F推导出的所有可能成立的函数依赖集合。

3.6 Armstrong公理

计算闭包的基本推导规则：

1. 自反律：$\beta \subseteq \alpha \Longrightarrow \alpha \to \beta$
2. 增补律：$\alpha \to \beta \Longrightarrow \gamma\alpha \to \gamma\beta$
3. 传递律：$\alpha \to \beta$且$\beta \to \gamma \Longrightarrow \alpha \to \gamma$

补充规则：

- 合并律：$\alpha \to \beta$且$\alpha \to \gamma \Longrightarrow \alpha \to \beta\gamma$
- 分解律：$\alpha \to \beta\gamma \Longrightarrow \alpha \to \beta$且$\alpha \to \gamma$
- 伪传递律：$\alpha \to \beta$且$\gamma\beta \to \delta \Longrightarrow \alpha\gamma \to \delta$

3.7 属性集闭包α^+

属性集闭包α^+是指由α函数决定的所有属性集合

用途：

- 判断α是否是SuperKey(若$\alpha^+ = R$)
- 判断$\alpha \to \beta$是否在F^+中成立(是否$\beta \subseteq \alpha^+$)

3.8 正则覆盖

目的：压缩函数依赖集，减少检查代价。

特点：最小的等价函数依赖集，去除冗余依赖和无关属性。

三种冗余情况：

1. 整个 FD 冗余→可由其他依赖推出。

2. 左边属性冗余。
3. 右边属性冗余。

计算过程：反复：

- 应用合并律。
- 找出无关属性，删除。
- 直到 F 不再变化。

4 分解

4.1 分解的目标

分解的目的是：

- 判断某个关系R是否处于“好” 的形式（Good form）：
  - 无冗余存储
  - 无插入/删除/更新异常
- 如果关系R不是 Good form，则分解为一组关系：$\{R_1, R_2, ..., R_n\}$

需要满足以下条件：

1. 无损连接分解（Lossless-join decomposition，必须保证）
2. 依赖保持（Dependency preservation，尽量保证）
3. 每个子关系R_i处于 Good form —— BCNF 或 3NF（优先满足）。

4.2 分解的良好性质

1.属性完整性

分解后，所有原始属性必须包含在子关系中：$R = R_1 \cup R_2$

2.无损连接分解

定义：对任意在R上的关系r，分解后通过自然连接$\bowtie$操作应当能恢复原始关系：$r = \pi_{R_1}(r) \bowtie \pi_{R_2}(r)$

判定条件(适用于两表分解)：$R_1 \cap R_2 \neq \phi$，且必须满足:$(R_1 \cap R_2) \rightarrow R_1$ or $(R_1 \cap R_2) \rightarrow R_2$

直观理解：两个子模式的公共属性如果是某一子关系的候选键，则保证无损连接。保证信息不丢失（lossless），这是分解时的强制性条件，必须满足。

3.依赖保持

定义：分解后，原有的函数依赖集 F，应尽量能在分解后的子关系R_i中直接检验，不需要回到全局 JOIN 后验证。这样可以避免在更新操作时，必须JOIN才能验证FD约束，提升效率。

具体做法：

- 对F中的每个依赖$\alpha \to \beta$，希望存在某个R_i包含$\alpha \cup \beta$，使得可以直接在R_i内部验证。
- 理想状态：$(F_1 \cup F_2 \cup ... \cup F_n)^+ = F^+$
- 注：如果依赖保持不满足，不是非法，但会影响更新代价。

```
result := a
while result change:
    for each Ri:
        t := (recult \cap Ri)^+ \cap Ri
        result := result \cup t
if result contains b, then a\to b is keeping
```

若 F 中所有依赖都通过上述方式被保持，说明该分解是 dependency preserving。

4.无冗余

希望分解后子关系处于BCNF 或 3NF，进一步减少冗余和异常。

一般来说，BCNF 保证最严格，但有时为了保持依赖，退而求其次用 3NF。

5 Boyce-Codd范式

5.1 BCNF定义

定义：关系模式R满足BCNF，针对一个函数依赖集F，如果对于F^+中的所有函数依赖$\alpha \to \beta$，都满足以下两个条件之一：

1. $\alpha \to \beta$是平凡依赖(即$\beta \subseteq \alpha$)
2. α是R的superkey，即$R \subseteq \alpha^+$或$\alpha \to R$

BCNF 更严格，要求所有的非平凡依赖，左边必须是超码，否则视为违反 BCNF。

5.2 如何测试BCNF

判断流程：对于每一个$\alpha \to \beta$:

1. 计算α^+(属性闭包)

2. 判断α^+是否包含所有R中的属性。若是，α^+是superkey；否则违反BCNF

简化测试：

- 通常只需检查F中的依赖是否违反BCNF
- 如果F中的都合法，则F^+中也不会违反BCNF

注意：检查分解后的子关系时，必须对F^+来判断是否满足 BCNF，而不能只看F

5.3 BCNF分解算法

```
result := {R}
done := false
compute F^+
while (not done) do
    if exists any Ri not satisfy BCNF:
        choose a non-trivial dependence a\to b:
            a is not superkey
        split Ri as:
            R1 := (a,b)
            R2 := Ri - b
        update result
    else
        done := true
```

最终 result 中所有子模式都是 BCNF，且分解是无损连接。

5.4 BCNF 与依赖保持

BCNF 分解不一定能保持依赖!

例子：$R = (J, K, L), F = \{JK \to L, L \to K\}$。候选键：JK,JL，R不满足BCNF。任意分解都会破坏$JK \to L$的保持性。

Lossless join，BCNF和dependency preservation三者不可兼得。所以实践中有时会使用 3NF 替代 BCNF，因为 3NF 可以保证依赖保持。

6 第三范式

6.1 引入动机

在某些情况下，BCNF 分解虽然可以消除冗余，但不能保证 **依赖保持**（Dependency Preservation）。而在实际数据库设计中，更新操作时需要高效地检查函数依赖是否被违反，若丢失依赖保持则必须进行昂贵的 JOIN 操作。

解决方案：引入一个更弱的范式——**第三范式（3NF）**，具备以下特点：

- 允许一定的冗余。
- 可以在单个子关系中独立检查 FD，**保证依赖保持**。
- **总是可以**进行无损连接、依赖保持的 3NF 分解。

6.2 定义

对于关系模式 R，若对于 F^+ 中的所有函数依赖 $\alpha \to \beta$，至少满足以下任一条件，则 R 满足 **3NF**：

1. $\alpha \to \beta$ 是平凡依赖（$\beta \subseteq \alpha$）。
2. α 是 R 的超码（superkey）。
3. $\beta - \alpha$ 中的每一个属性 A，都是 R 的某个候选键中的主属性。

关系：BCNF $\Rightarrow$ 3NF。3NF 是对 BCNF 的**最小放宽**，用于保证依赖保持。

6.3 示例

关系 $R = (J, K, L)$，函数依赖集 $F = \{JK \to L, L \to K\}$。

- 候选键：JK、JL。
- $JK \to L$：JK 是超码，满足。
- $L \to K$：K 是候选键中的一部分，满足。

因此，R 满足 3NF。但如果分解成 BCNF，$JK \to L$ 不会被保留，导致依赖无法直接检查，需要 JOIN。

6.4 3NF 中的冗余

3NF 允许存在冗余，这正是换取依赖保持的代价。

示例 $R(J, K, L)$ 中，$L \to K$ 导致 (L, K) 对重复存储，甚至出现 NULL 值。

总结：

- BCNF：彻底消除冗余，但可能丢失依赖保持。
- 3NF：允许一定冗余，保证依赖保持。

6.5 3NF 检测方法

- 只需检查 F 中的 FD，不需要检查 F^+。
- 对每个 $\alpha \to \beta$：
  - 计算 α^+，判断 α 是否是超码。
  - 如果不是，检查 $\beta - \alpha$ 中每个属性是否为候选键中的主属性。

注意：

- 需要找出所有候选键，计算候选键是 NP-hard 问题。

6.6 3NF 分解算法

1. 对 F 求出规范覆盖 F_c。
2. 对 F_c 中每个 $\alpha \to \beta$，若当前已有子关系中无 $\alpha\beta$，则新建子关系 $R_i = (\alpha, \beta)$。
3. 最后，若当前子关系中无包含任何候选键的关系，补充一个包含候选键的子关系 R_{key}。

保证：

- Lossless-join。
- Dependency-preserving。

6.7 示例

关系模式：

```
Banker-info-schema = (branch-name, customer-name, banker-name, office-number)
```

函数依赖集：

$F = \{banker\text{-}name \to branch\text{-}name, office\text{-}number;\ customer\text{-}name,\ branch\text{-}name \to banker-name\}$

候选键：

$\{customer-name, branch-name\}$

分解结果：

```
Banker-office-schema = (banker-name, branch-name, office-number)
Banker-schema = (customer-name, branch-name, banker-name)
```

分解后，$Banker-schema$ 包含候选键，分解结束。

6.8 3NF 与 BCNF 的比较

方面	BCNF	3NF
是否 Always Lossless-join	是	是
是否 Always Dependency-preserving	否	是
是否可能有冗余	否	有可能

Table 1. BCNF 与 3NF 的比较

6.9 设计目标总结

设计目标优先级：

- BCNF。
- Lossless join。
- Dependency preservation。

如果不能全部达成，通常选择：

- 接受 3NF（允许冗余，保证依赖保持）。
- 或接受 BCNF（消除冗余，牺牲依赖保持）。

6.10 跨关系检测 FD

若分解未保持依赖，可用 **物化视图（Materialized View）** 辅助检查：

- 物化视图定义为 $\alpha\beta$ 上的投影。
- 视图可定义 α 为视图上的候选键。
- 检查 $\alpha \to \beta$ 变成检查 α 是否为视图的候选键，效率更高。

代价：

- 空间代价：需要存储物化视图。
- 时间代价：视图需要及时更新。
- 部分数据库系统不支持视图上的候选键声明。

7 多值依赖（Multivalued Dependencies）

7.1 概述

- 即使在 BCNF 中，有些数据库模式仍然不够规范化。

- 示例数据库：`classes(course, teacher, book)`，其中 $(c, t, b) \in$ classes 表示 t 教授课程 c，b 是 c 课程所需教材。
- 每门课程 *course* 对应一组教师 *teacher*（$1:n$）以及一组教材 *book*（$1:n$），教师和教材是独立的多值属性。
- **多值属性**之间相互独立，应该避免交叉组合造成冗余。

7.2 示例数据

course	teacher	book
database	Avi	DB Concepts
database	Avi	DB system (Ullman)
database	Hank	DB Concepts
database	Hank	DB system (Ullman)
database	Sudarshan	DB Concepts
database	Sudarshan	DB system (Ullman)
operating systems	Avi	OS Concepts
operating systems	Avi	OS system (Shaw)
operating systems	Jim	OS Concepts
operating systems	Jim	OS system (Shaw)

- 目前 `classes` 关系中只有 trivial FDs，因此该关系在 BCNF 中。
- 但存在 **冗余**和 **插入异常**，例如添加新的教师，需要插入 *course* 和每本 *book* 的组合。

7.3 关系分解

- 为避免冗余，推荐将 `classes` 分解为两个关系：

teaches

course	teacher
database	Avi
database	Hank
database	Sudarshan
operating systems	Avi
operating systems	Jim

text

course	book
database	DB Concepts
database	DB system (Ullman)
operating systems	OS Concepts
operating systems	OS system (Shaw)

- 这样可以保证两个关系分别满足 Fourth Normal Form (4NF)。

7.4 多值依赖的定义

- 设 R 是关系模式，$\alpha, \beta \subseteq R$，若对 R 的任意关系 r，所有满足 $t_1[\alpha] = t_2[\alpha]$ 的元组对 t_1, t_2，存在 t_3, t_4 满足以下条件：

$$
\begin{aligned}
t_3[\alpha] = t_4[\alpha] = t_1[\alpha] = t_2[\alpha] \\
t_3[\beta] = t_1[\beta], \quad t_4[\beta] = t_2[\beta] \\
t_3[R - \alpha - \beta] = t_2[R - \alpha - \beta] \\
t_4[R - \alpha - \beta] = t_1[R - \alpha - \beta]
\end{aligned}
$$

- 记作 $\alpha \twoheadrightarrow \beta$，即 α 多值决定 β。
- 若 $\beta \subseteq \alpha$ 或 $\alpha \cup \beta = R$，则 $\alpha \twoheadrightarrow \beta$ 是平凡的。

7.5 多值依赖的理论性质

- **推导规则：**
  - 若 $\alpha \to \beta$，则 $\alpha \twoheadrightarrow \beta$（即函数依赖是特殊的多值依赖）。
- 闭包 D^+ 是由所有逻辑推导出的 **函数依赖** 和 **多值依赖** 的集合。
- 对于简单的多值依赖，可以直接用定义推理。对于复杂依赖，需借助形式推导系统。

7.6 补充示例

Employee-name	project	dependent
smith	p1	tom
smith	p2	anna
smith	p1	anna
smith	p2	tom
julia	p3	tom
julia	p3	mary
julia	p1	tom
julia	p1	mary

- Key = {employee-name, project, dependent}。
- **多值依赖**的典型场景：employee 对 project 和 dependent 各自具有独立的多值依赖关系。

7.7 总结

- 多值依赖揭示了 BCNF 中可能未充分规范化的场景。

- 通过分解，减少冗余和插入/删除异常。
- 目标是使关系满足 Fourth Normal Form (4NF)。

8 第四范式

8.1 第四范式（4NF）定义

一个关系模式 R 关于一组函数依赖和多值依赖 D，若对于 D^+ 中所有形如 $\alpha \twoheadrightarrow \beta$ 的多值依赖，$\alpha \subseteq R, \beta \subseteq R$，满足以下至少一个条件，则称 R 满足第四范式（4NF）：

- $\alpha \twoheadrightarrow \beta$ 是平凡的（trivial）：
  - $\beta \subseteq \alpha$ 或
  - $\alpha \cup \beta = R$
- α 是关系 R 的一个超码（superkey）。

性质：

- 如果一个关系在 4NF 中，它一定在 BCNF 中。

8.2 为什么需要4NF？

- 有些在 BCNF 中的关系模式仍存在**多值依赖**（Multivalued Dependencies, MVDs）导致的冗余和异常。
- 多值依赖的典型特征是某些属性组之间的值是独立的。

例子：

- 关系 `classes(course, teacher, book)`，对于每门课程 `course`，有一组 `teacher` 和一组 `book`，且两者相互独立：
  - $course \twoheadrightarrow teacher$
  - $course \twoheadrightarrow book$

问题：

- 数据冗余（Redundant）：
  - 插入新教师需要复制教材信息。
  - 删除一个教师，可能导致教材信息丢失。
- 插入/删除异常。

8.3 4NF 分解要求

- 将关系 R 分解成 $R_1, R_2, \ldots, R_n$，使每个 R_i 满足 4NF。
- 分解过程应保持 **Lossless Join**（无损连接）。

对子模式 R_i 应保留的依赖：

- D_i 包含以下依赖：
  - 所有只涉及 R_i 属性的函数依赖。
  - 所有形如 $\alpha \twoheadrightarrow (\beta \cap R_i)$ 的多值依赖，且 $\alpha \subseteq R_i$。

8.4 4NF 分解算法

1. 初始 $result := \{R\}$，$done := false$。
2. 计算 D^+。
3. 设 D_i 为 D^+ 限制在 R_i 上的依赖。
4. **循环：**
   - 若 $result$ 中存在不满足 4NF 的 R_i，则执行以下步骤：
     (a) 取出 R_i 上一个非平凡 MVD $\alpha \twoheadrightarrow \beta$。
     (b) 执行分解：
     $$result := (result - R_i) \cup (\alpha, \beta) \cup (R_i - \beta)$$
5. 直到所有 R_i 满足 4NF。

保证：分解后，每个 R_i 在 4NF 中，且分解是 Lossless Join。

8.5 进一步的范式（Further Normal Forms）

- **连接依赖**（Join Dependencies, JD）推广了多值依赖。
  - 对应 **第五范式**（5NF），也称 **投影-连接范式**（PJNF）。
- 还有更一般的约束，称为 **域-码范式**（Domain-Key Normal Form, DKNF）。

问题：

- DKNF 推理困难。
- 没有一套可靠、完备的推理规则。
- 实际应用中极少采用。

8.6 小结

范式	目标
BCNF	消除所有非平凡 FD 依赖导致的数据冗余
4NF	消除多值依赖（MVD）导致的数据冗余
5NF (PJNF)	消除连接依赖导致的数据冗余

- 4NF 是 BCNF 的增强。
- 设计复杂数据库时若存在独立的多值属性，应考虑 4NF 分解。