数据库规范化形式与实例详解

数据库规范化（Normalization）是数据库设计中的一种方法，用于**消除冗余数据**、**提高数据一致性**，并**避免数据更新时的异常**。通过逐步应用不同的规范化标准（范式），可以将数据库关系划分为结构更简单、依赖更明确的关系。以下将详细介绍从**第一范式（1NF）到Boyce-Codd范式（BCNF）**的定义与具体实例。

正规化范式 (Normal Forms)

1. **1NF (第一范式)**：所有属性的值必须是原子的，即不可再分。
2. **2NF (第二范式)**：非主属性不能对任何候选键部分依赖。
3. **3NF (第三范式)**：对于所有非平凡的函数依赖 (X \to A)，要么 (X) 是超键，要么 (A) 是主属性（没有传递依赖）。
4. **BCNF (Boyce-Codd 正规形)**：对于所有非平凡的函数依赖 (X \to A)，(X) 必须是超键。

1. 第一范式（1NF）

- **定义**：关系中的每个属性值必须是**原子的（Atomic）**，即每个属性的值都是不可分割的，不能包含集合、列表或嵌套关系。
- **实例**： 假设有一个关系 `CRS_PREF`，用于表示教授对不同课程的偏好：

Prof	Course	Fac_Dept	Crs_Dept
Smith	353	Comp Sci	Comp Sci
Smith	379	Comp Sci	Comp Sci
Turner	456	Chemistry	Mathematics

在这个表中，**每个单元格的值都是原子的**，没有多值或复杂类型，这意味着它满足 1NF。然而，这种形式存在一些问题：

  - **数据重复**：例如教授 Smith 和课程 353、379 的关联会导致部门信息多次重复。
  - **插入异常**：如果我们想插入一个新教授，但他还没有教授任何课程，那么我们无法插入记录，因为缺少课程信息。
  - **删除异常**：删除某个教授的最后一门课程可能导致该教授的部门信息也被删除。

2. 第二范式（2NF）

- **定义**：一个关系模式 R 处于 2NF（Second Normal Form），当且仅当 R 的每个非主属性完全依赖于 R 的**每个候选键（Candidate Key）**。这里，依赖的左边是候选键，右边是非主属性。
  - **部分函数依赖（Partial Functional Dependency）**：如果存在一个非主属性仅依赖于候选键的某个子集，则称其存在部分函数依赖。这里，依赖的左边是候选键的子集，右边是非主属性。这种情况下，该关系不符合 2NF，需要分解。
- **实例**： 在上面的 `CRS_PREF` 表中，`Fac_Dept` 只依赖于 `Prof`，而与 `Course` 无关。这意味着存在**部分依赖**，导致数据的重复存储。为了解决这个问题，我们可以将 `CRS_PREF` 分解为以下三个关系：
  1. **COURSE_PREF**（教授与课程）：

Prof	Course
Smith	353
Smith	379
Turner	456

2. **COURSE**（课程与部门）：

Course	Dept
353	Comp Sci
379	Comp Sci
456	Mathematics

3. **FACULTY**（教授与部门）：

Prof	Dept
Smith	Comp Sci
Turner	Chemistry

通过这样的分解，消除了部分依赖（Partial Dependency），关系模式进入了 2NF，减少了数据冗余。

3. 第三范式（3NF）

- **定义**：一个关系模式 R 处于 3NF（Third Normal Form），当且仅当对于每一个非平凡的函数依赖（Functional Dependency） X → A，要么 X 是**超键（Superkey）**，要么 A 是**候选键的属性。**
  - **传递函数依赖（Transitive Dependency）**：如果一个属性通过另一个属性间接依赖于候选键，则存在传递依赖。这会导致数据冗余和更新异常，需要消除。
- **实例**： 考虑以下 `TEACHES` 表：

Course	Prof	Room	Room_Cap	Enrol_Lmt
353	Smith	A532	45	40
456	Turner	B278	50	45

在这个关系中，`Room_Cap` 是通过 `Room` 间接依赖于 `Course`，这是一种**传递依赖**。为了消除这种依赖，我们可以将 `TEACHES` 表分解为两个子关系：

1. **COURSE_DETAILS**（课程、教授和房间）：

Course	Prof	Room
353	Smith	A532
456	Turner	B278

2. **ROOM_DETAILS**（房间、容量和注册限制）：

Room	Room_Cap	Enrol_Lmt
A532	45	40
B278	50	45

通过分解，我们消除了传递依赖，使得每个非主属性直接依赖于候选键，从而达到了 3NF。

4. Boyce-Codd 范式（BCNF）

- **定义**：一个关系模式处于 BCNF（Boyce-Codd Normal Form），当且仅当对于每一个非平凡的函数依赖 X → A，X 必须是**超键（Superkey）**。
    - BCNF 是 3NF（Third Normal Form）的强化版本，进一步消除了由非超键决定的属性带来的冗余。
- **实例**：假设有一个关系 `BOOKING`：

Title	Theater	City
MovieA	Cineplex	NYC
MovieB	Cineplex	NYC

在这个表中，`Theater` → `City`，但 `Theater` 不是超键，因此该关系不符合 BCNF。为了解决这个问题，我们可以将其分解为两个关系：

1. **THEATER_DETAILS**（影院和城市）：

Theater	City
Cineplex	NYC

2. **MOVIE_SHOWING**（电影和影院）：

Title	Theater
MovieA	Cineplex
MovieB	Cineplex

通过这样的分解，所有属性都由超键唯一决定，从而消除了冗余，使关系达到 BCNF。

5. 总结与判定示例

- **判断一个关系模式 R 最高符合的范式（从高到低判断）**：
    1. **BCNF 检查**：
        - 判断关系中的每一个非平凡函数依赖 `X → A`，是否 `X` 是超键（Superkey）。如果存在任意非平凡依赖的左边 `X` 不是超键，则该关系不符合 BCNF，继续检查是否符合 3NF。
    2. **3NF 检查**：
        - 判断关系中的每一个非平凡函数依赖 `X → A`，是否满足以下条件之一：

        - `X` 是超键。
        - `A` 是候选键的属性。
    - 如果存在传递依赖（Transitive Dependency），即非主属性通过另一个非主属性间接依赖于主键，则关系不符合 3NF，继续检查是否符合 2NF。
3. **2NF 检查**：
    - 首先确认关系模式符合 1NF。
    - 检查每一个非主属性是否完全依赖于整个候选键。如果存在部分依赖（Partial Dependency），即非主属性只依赖于候选键的某个子集，则关系不符合 2NF，继续判断是否符合 1NF。
4. **1NF 检查**：
    - 检查关系中的每个属性值是否都是原子的，即不可再分的基本类型。如果某个属性值包含集合、列表或嵌套关系，则不符合 1NF。

实用性提示：

- **从高到低逐步判断**的方式更有效，因为一旦关系不符合某个高范式，它必然不符合更高的范式。因此从 BCNF 开始逐步降低，可以更快速确定最高符合的范式。

例题： 假设有一个关系 `STUDENT_COURSE`：

StudentID	CourseID	Instructor	Dept
1	C1	Prof. A	CS
2	C1	Prof. A	CS
3	C2	Prof. B	Math

- **BCNF 检查**：
    - 依赖 `Instructor → Dept` 中，`Instructor` 不是超键，因此不符合 BCNF。
- **3NF 检查**：
    - 依赖 `Instructor → Dept` 是传递依赖，因为 `Dept` 通过 `Instructor` 间接依赖于 `(StudentID, CourseID)`，因此不符合 3NF。
- **2NF 检查**：
    - `Dept` 对 `Instructor` 存在部分依赖，因为它不依赖于整个候选键 `(StudentID, CourseID)`，因此不符合 2NF。
- **1NF 检查**：
    - 该关系符合 1NF，因为每个属性值都是原子的。

通过分析，我们可以得出该关系模式只符合 **1NF**。

- **判断一个关系模式 R 最高符合的范式**：
    - 检查 R 是否符合 1NF：每个属性值是否都是原子的。
    - 检查 R 是否符合 2NF：是否存在部分函数依赖，如果有则不符合 2NF。
    - 检查 R 是否符合 3NF：是否存在传递依赖，如果有则不符合 3NF。
    - 检查 R 是否符合 BCNF：是否所有非平凡的函数依赖的左边都是超键，如果不是，则不符合 BCNF。

分解 (Decomposition)

- **分解的属性保留条件**：在分解中，所有属性必须在分解后的关系中保留。
- **分解的两个重要性质**：
    1. **依赖保持性**：所有的原始函数依赖都能在分解后的关系中表达。
    2. **无损连接性质**：在自然连接操作后，分解后的关系能重建原始关系。

依赖保持 (Dependency Preserving)

- 分解 (D = {R1, ..., Rn}) 被称为依赖保持的，如果所有投影后的函数依赖的闭包等于原始函数依赖的闭包，即 ((F1 \cup ... \cup Fn)^+ = F^+)。

无损连接性质 (Lossless Join Property)

- 若分解 (D = {R1, ..., Rm}) 满足无损连接性质，则对于任意满足函数依赖的关系实例，所有分解的自然连接的结果应与原始关系相同。
- 检查方法：分解为 (R1, R2) 的无损连接性质成立，当且仅当交集 (R1 \cap R2) 构成 (R1) 或 (R2) 的一个超键。

BCNF 和 3NF 的分解算法

- **BCNF 分解算法 (TO_BCNF)**：
    - 找到违反 BCNF 的函数依赖并分解关系，直到所有关系都满足 BCNF。
    - 该分解确保无损连接，但可能不保持依赖。
- **3NF 分解算法**：
    - 通过最小覆盖得到函数依赖的最简形式。
    - 创建包含候选键的关系以确保依赖保持，并进行无损连接。
    - 3NF 分解总是能够找到依赖保持且无损的分解，但可能保留一定的冗余。

最小覆盖 (Minimal Cover)

- **最小覆盖**：函数依赖的最小集合，具有相同的闭包且去除了冗余。
- 计算最小覆盖的步骤：
    1. **右侧简化**：将右侧多个属性拆分为多个单属性的函数依赖。
    2. **左侧简化**：去除左侧的冗余属性。
    3. **冗余去除**：删除冗余的函数依赖。