数据库设计三大范式应用实例剖析

引言

数据库的设计范式是数据库设计所需要满足的规范，满足这些规范的数据库是简洁的、结构明晰的，同时，不会发生插入（insert）、删除（delete）和更新（update）操作异常。反之则是乱七八糟，不仅给数据库的编程人员制造麻烦，而且面目可憎，可能存储了大量不需要的冗余信息。

设计范式是不是很难懂呢？非也，大学教材上给我们一堆数学公式我们当然看不懂，也记不住。所以我们很多人就根本不按照范式来设计数据库。

实质上，设计范式用很形象、很简洁的话语就能说清楚，道明白。本文将对范式进行通俗地说明，并以笔者曾经设计的一个简单论坛的数据库为例来讲解怎样将这些范式应用于实际工程。

范式说明

第一范式（1NF）：数据库表中的字段都是单一属性的，不可再分。这个单一属性由基本类型构成，包括整型、实数、字符型、逻辑型、日期型等。

例如，如下的数据库表是符合第一范式的：

字段1	字段2	字段3	字段4

而这样的数据库表是不符合第一范式的：

字段1	字段2	字段3		字段4
		字段3.1	字段3.2	

很显然，在当前的任何关系数据库管理系统（DBMS）中，傻瓜也不可能做出不符合第一范式的数据库，因为这些 DBMS 不允许你把数据库表的一列再分成二列或多列。因此，你想在现有的 DBMS 中设计出不符合第一范式的数据库都是不可能的。

第二范式（2NF）：数据库表中不存在非关键字段对任一候选关键字段的部分函数依赖（部分函数依赖指的是存在组合关键字中的某些字段决定非关键字段的情况），也即所有非关键字

段都完全依赖于任意一组候选关键字。

假定选课关系表为 SelectCourse(学号, 姓名, 年龄, 课程名称, 成绩, 学分)，关键字为组合关键字(学号, 课程名称)，因为存在如下决定关系：

(学号, 课程名称) → (姓名, 年龄, 成绩, 学分)

这个数据库表不满足第二范式，因为存在如下决定关系：

(课程名称) → (学分)

(学号) → (姓名, 年龄)

即存在组合关键字中的字段决定非关键字的情况。

由于不符合2NF，这个选课关系表会存在如下问题：

(1) 数据冗余：

同一门课程由 n 个学生选修，"学分"就重复 n-1次；同一个学生选修了 m 门课程，姓名和年龄就重复了 m-1次。

(2) 更新异常：

若调整了某门课程的学分，数据表中所有行的"学分"值都要更新，否则会出现同一门课程学分不同的情况。

(3) 插入异常：

假设要开设一门新的课程，暂时还没有人选修。这样，由于还没有"学号"关键字，课程名称和学分也无法记录入数据库。

(4) 删除异常：

假设一批学生已经完成课程的选修，这些选修记录就应该从数据库表中删除。但是，与此同时，课程名称和学分信息也被删除了。很显然，这也会导致插入异常。

把选课关系表 SelectCourse 改为如下三个表：

学生：Student(学号, 姓名, 年龄)；

课程：Course(课程名称, 学分)；

选课关系：SelectCourse(学号, 课程名称, 成绩)。

这样的数据库表是符合第二范式的，消除了数据冗余、更新异常、插入异常和删除异常。

另外，所有单关键字的数据库表都符合第二范式，因为不可能存在组合关键字。

第三范式（3NF）：在第二范式的基础上，数据表中如果不存在非关键字段对任一候选关键字段的传递函数依赖则符合第三范式。所谓传递函数依赖，指的是如果存在"A → B → C"的决定关系，则 C 传递函数依赖于 A。因此，满足第三范式的数据库表应该不存在如下依赖关系：

关键字段 → 非关键字段 x → 非关键字段 y

假定学生关系表为 Student(学号，姓名，年龄，所在学院，学院地点，学院电话)，关键字为单一关键字“学号”，因为存在如下决定关系：

(学号)→(姓名，年龄，所在学院，学院地点，学院电话)

这个数据库是符合2NF 的，但是不符合3NF，因为存在如下决定关系：

(学号)→(所在学院)→(学院地点，学院电话)

即存在非关键字段“学院地点”、“学院电话”对关键字段“学号”的传递函数依赖。

它也会存在数据冗余、更新异常、插入异常和删除异常的情况，读者可自行分析得知。

把学生关系表分为如下两个表：

学生：(学号，姓名，年龄，所在学院)；

学院：(学院，地点，电话)。

这样的数据库表是符合第三范式的，消除了数据冗余、更新异常、插入异常和删除异常。

鲍依斯-科得范式（BCNF)：在第三范式的基础上，数据库表中如果不存在任何字段对任一候选关键字段的传递函数依赖则符合第三范式。

假设仓库管理关系表为 StorehouseManage(仓库 ID, 存储物品 ID, 管理员 ID, 数量)，且有一个管理员只在一个仓库工作；一个仓库可以存储多种物品。这个数据库表中存在如下决定关系：

(仓库 ID, 存储物品 ID) →(管理员 ID, 数量)

(管理员 ID, 存储物品 ID) → (仓库 ID, 数量)

所以，(仓库 ID, 存储物品 ID)和(管理员 ID, 存储物品 ID)都是 StorehouseManage 的候选关键字，表中的唯一非关键字段为数量，它是符合第三范式的。但是，由于存在如下决定关系：

(仓库 ID) → (管理员 ID)

(管理员 ID) → (仓库 ID)

即存在关键字段决定关键字段的情况，所以其不符合 BCNF 范式。它会出现如下异常情况：

(1) 删除异常：

当仓库被清空后，所有"存储物品 ID"和"数量"信息被删除的同时，"仓库 ID"和"管理员 ID"信息也被删除了。

(2) 插入异常：

当仓库没有存储任何物品时，无法给仓库分配管理员。

(3) 更新异常：

如果仓库换了管理员，则表中所有行的管理员 ID 都要修改。

把仓库管理关系表分解为二个关系表：

仓库管理：StorehouseManage(仓库 ID, 管理员 ID)；

仓库：Storehouse(仓库 ID, 存储物品 ID, 数量)。

这样的数据库表是符合 BCNF 范式的，消除了删除异常、插入异常和更新异常。

范式应用

我们来逐步搞定一个论坛的数据库，有如下信息：

（1） 用户：用户名，email，主页，电话，联系地址

（2） 帖子：发帖标题，发帖内容，回复标题，回复内容

第一次我们将数据库设计为仅仅存在表：

用户名	email	主页	电话	联系地址	发帖标题	发帖内容	回复标题	回复内容

这个数据库表符合第一范式，但是没有任何一组候选关键字能决定数据库表的整行，唯一的关键字段用户名也不能完全决定整个元组。我们需要增加"发帖 ID"、"回复 ID"字段，即将表修改为：

用户名	email	主页	电话	联系地址	发帖ID	发帖标题	发帖内容	回复 ID	回复标题	回复内容

这样数据表中的关键字(用户名，发帖 ID，回复 ID)能决定整行：
(用户名,发帖 ID,回复 ID) → (email,主页,电话,联系地址,发帖标题,发帖内容,回复标题,回复内容)
但是，这样的设计不符合第二范式，因为存在如下决定关系：
(用户名) → (email,主页,电话,联系地址)
(发帖 ID) → (发帖标题,发帖内容)
(回复 ID) → (回复标题,回复内容)
即非关键字段部分函数依赖于候选关键字段，很明显，这个设计会导致大量的数据冗余和操作异常。

我们将数据库表分解为（带下划线的为关键字）：

（1） 用户信息：用户名，email，主页，电话，联系地址

（2） 帖子信息：发帖 ID，标题，内容

（3） 回复信息：回复 ID，标题，内容

（4） 发贴：用户名，发帖 ID

（5） 回复：发帖 ID，回复 ID

这样的设计是满足第1、2、3范式和 BCNF 范式要求的，但是这样的设计是不是最好的呢？不一定。

观察可知，第4项“发帖”中的“用户名”和“发帖 ID”之间是1：N 的关系，因此我们可以

把“发帖”合并到第2项的“帖子信息”中；第5项“回复”中的“发帖 ID”和“回复 ID”之间也是1：N 的关系，因此我们可以把“回复”合并到第3项的“回复信息”中。这样可以一定量地减少数据冗余，新的设计为：

（1） 用户信息：用户名，email，主页，电话，联系地址

（2） 帖子信息：用户名，发帖 ID，标题，内容

（3） 回复信息：发帖 ID，回复 ID，标题，内容

数据库表1显然满足所有范式的要求；

数据库表2中存在非关键字段“标题”、“内容”对关键字段“发帖 ID”的部分函数依赖，即不满足第二范式的要求，但是这一设计并不会导致数据冗余和操作异常；

数据库表3中也存在非关键字段“标题”、“内容”对关键字段“回复 ID”的部分函数依赖，也不满足第二范式的要求，但是与数据库表2相似，这一设计也不会导致数据冗余和操作异常。

由此可以看出，并不一定要强行满足范式的要求，对于1：N 关系，当1的一边合并到 N 的那边后，N 的那边就不再满足第二范式了，但是这种设计反而比较好！

对于 M：N 的关系，不能将 M 一边或 N 一边合并到另一边去，这样会导致不符合范式要求，同时导致操作异常和数据冗余。

对于1：1的关系，我们可以将左边的1或者右边的1合并到另一边去，设计导致不符合范式要求，但是并不会导致操作异常和数据冗余。

结论

满足范式要求的数据库设计是结构清晰的，同时可避免数据冗余和操作异常。这并意味着不符合范式要求的设计一定是错误的，在数据库表中存在1：1或1：N 关系这种较特殊的情况下，合并导致的不符合范式要求反而是合理的。

在我们设计数据库的时候，一定要时刻考虑范式的要求。