数据库范式——通俗易懂【转】

(2012-04-02 21:15:43)

数据库范式是数据库设计中必不可少的知识，没有对范式的理解，就无法设计出高效率、优雅的数据库。甚至设计出错误的数据库。而想要理解并掌握范式却并不是那么容易。教科书中一般以关系代数的方法来解释数据库范式。这样做虽然能够十分准确的表达数据库范式，但比较抽象，不太直观，不便于理解，更难以记忆。

一、基础概念

实体：现实世界中客观存在并可以被区别的事物。比如“一个学生”、“一本书”、“一门课”等等。值得强调的是这里所说的“事物”不仅仅是看得见摸得着的“东西”，它也可以是虚拟的，不如说“老师与学校的关系”。

- 属性：教科书上解释为：“实体所具有的某一特性”，由此可见，属性一开始是个逻辑概念，比如说，“性别”是“人”的一个属性。在关系数据库中，属性又是个物理概念，属性可以看作是“表的一列”。
- 元组：表中的一行就是一个元组。
- 分量：元组的某个属性值。在一个关系数据库中，它是一个操作原子，即关系数据库在做任何操作的时候，属性是“不可分的”。否则就不是关系数据库了。
- 码：表中可以唯一确定一个元组的某个属性（或者属性组），如果这样的码有不止一个，那么大家都叫候选码，我们从候选码中挑一个出来做老大，它就叫主码。
- 全码：如果一个码包含了所有的属性，这个码就是全码。
- 主属性：一个属性只要在任何一个候选码中出现过，这个属性就是主属性。
- 非主属性：与上面相反，没有在任何候选码中出现过，这个属性就是非主属性。
- 外码：一个属性（或属性组），它不是码，但是它别的表的码，它就是外码。

二、6 个范式

好了，上面已经介绍了我们掌握范式所需要的全部基础概念，下面我们就来讲范式。首先要明白，范式的包含关系。一个数据库设计如果符合第二范式，一定也符合第一范式。如果符合第三范式，一定也符合第二范式…

第一范式（1NF）：属性不可分。

在前面我们已经介绍了属性值的概念，我们说，它是“不可分的”。而第一范式要求属性也不可分。那么它和属性值不可分有什么区别呢？给一个例子：

name	tel	age
大宝	13612345678	22
小明	13988776655	010—1234567	21

Ps：这个表中，属性值“分”了。

name	tel		age
	手机	座机	
大宝	13612345678	021—9876543	22
小明	13988776655	010—1234567	21

Ps：这个表中，属性 “分”了。

这两种情况都不满足第一范式。不满足第一范式的数据库，不是关系数据库！所以，我们在任何关系数据库管理系统中，做不出这样的“表”来。

第二范式（2NF）：符合 1NF，并且，非主属性完全依赖于码。

听起来好像很神秘，其实真的没什么。

一个候选码中的主属性也可能是好几个。如果一个主属性，它不能单独做为一个候选码，那么它也不能确定任何一个非主属性。给一个反例：我们考虑一个小学的教务管理系统，学生上课指定一个老师，一本教材，一个教室，一个时间，大家都上课去吧，没有问题。那么数据库怎么设计？（学生上课表）

学生	课程	老师	老师职称	教材	教室	上课时间
小明	一年级语文（上）	大宝	副教授	《小学语文1》	101	14：30

一个学生上一门课，一定在特定某个教室。所以有（学生，课程）—>教室

一个学生上一门课，一定是特定某个老师教。所以有（学生，课程）—>老师

一个学生上一门课，他老师的职称可以确定。所以有（学生，课程）—>老师职称

一个学生上一门课，一定是特定某个教材。所以有（学生，课程）—>教材

一个学生上一门课，一定在特定时间。所以有（学生，课程）—>上课时间

因此（学生，课程）是一个码。

然而，一个课程，一定指定了某个教材，一年级语文肯定用的是《小学语文 1》，那么就有课程—>教材。（学生，课程）是个码，课程却决定了教材，这就叫做不完全依赖，或者说部分依赖。出现这样的情况，就不满足第二范式！

有什么不好吗？你可以想想：

1、校长要新增加一门课程叫“微积分”，教材是《大学数学》，怎么办？学生还没选课，而学生又是主属性，主属性不能空，课程怎么记录呢，教材记到哪呢？……郁闷了吧?(插入异常)

2、下学期没学生学一年级语文（上）了，学一年级语文（下）去了，那么表中将不存在一年级语文（上），也就没了《小学语文 1》。这时候，校长问：一年级语文（上）用的什么教材啊？……郁闷了吧?(删除异常)

3、校长说：一年级语文（上）换教材，换成《大学语文》。有 10000 个学生选了这么课，改动好大啊！改累死了……郁闷了吧？（修改异常）

那应该怎么解决呢？投影分解，将一个表分解成两个或若干个表

学生	课程	老师	老师职称	教室	上课时间
小明	一年级语文（上）	大宝	副教授	101	14：30

学生上课表新

课程	教材
一年级语文（上）	《小学语文 1》

课程的表　　第三范式（3NF）：符合 2NF，并且，消除传递依赖

上面的“学生上课表新”符合 2NF，可以这样验证：两个主属性单独使用，不用确定其它四个非主属性的任何一个。但是它有传递依赖！

在哪呢？问题就出在“老师”和“老师职称”这里。一个老师一定能确定一个老师职称。

有什么问题吗？想想：

1、老师升级了，变教授了，要改数据库，表中有 N 条，改了 N 次……（修改异常）

2、没人选这个老师的课了，老师的职称也没了记录……（删除异常）

3、新来一个老师，还没分配教什么课，他的职称记到哪？……（插入异常）

那应该怎么解决呢？和上面一样，投影分解：

学生	课程	老师	教室	上课时间
小明	一年级语文（上）	大宝	101	14：30

老师	老师职称
大宝	副教授

BC 范式（BCNF）：符合 3NF，并且，主属性不依赖于主属性

若关系模式属于第一范式，且每个属性都不传递依赖于键码，则 R 属于 BC 范式。

通常

BC 范式的条件有多种等价的表述：每个非平凡依赖的左边必须包含键码；每个决定因素必须包含键码。

BC 范式既检查非主属性，又检查主属性。当只检查非主属性时，就成了第三范式。满足 BC 范式的关系都必然满足第三范式。

还可以这么说：若一个关系达到了第三范式，并且它只有一个候选码，或者它的每个候选码都是单属性，则该关系自然达到 BC 范式。

一般，一个数据库设计符合 3NF 或 BCNF 就可以了。在 BC 范式以上还有第四范式、第五范式。

第四范式：要求把同一表内的多对多关系删除。

第五范式：从最终结构重新建立原始结构。

但在绝大多数应用中不需要设计到这种程度。并且，某些情况下，过于范式化甚至会对数据库的逻辑可读性和使用效率起到阻碍。数据库中一定程度的冗余并不一定是坏事情。如果你对第四范式、第五范式感兴趣可以看一看专业教材，从头学起，并且忘记我说的一切，以免对你产生误导

数据库设计中的一些技巧

1. 原始单据与实体之间的关系

可以是一对一、一对多、多对多的关系。在一般情况下，它们是一对一的关系：即一张原始单据对应且只对应一个实体。在特殊情况下，它们可能是一对多或多对一的关系，即一张原始单证对应多个实体，或多张原始单证对应一个实体。这里的实体可以理解为基本表。明确这种对应关系后，对我们设计录入界面大有好处。

〖例 1〗：一份员工履历资料，在人力资源信息系统中，就对应三个基本表：员工基本情况表、社会关系表、工作简历表。这就是“一张原始单证对应多个实体”的典型例子。

2. 主键与外键

一般而言，一个实体不能既无主键又无外键。在 E—R 图中，处于叶子部位的实体，可以定义主键，也可以不定义主键(因为它无子孙)，但必须要有外键(因为它有父亲)。

主键与外键的设计，在全局数据库的设计中，占有重要地位。当全局数据库的设计完成以后，有个美国数据库设计专家说：“键，到处都是键，除了键之外，什么也没有”，这就是他的数据库设计经验之谈，也反映了他对信息系统核心(数据模型)的高度抽象思想。因为：主键是实体的高度抽象，主键与外键的配对，表示实体之间的连接。

3. 基本表的性质

基本表与中间表、临时表不同，因为它具有如下四个特性：

(1) 原子性。基本表中的字段是不可再分解的。

(2) 原始性。基本表中的记录是原始数据（基础数据）的记录。

(3) 演绎性。由基本表与代码表中的数据，可以派生出所有的输出数据。

(4) 稳定性。基本表的结构是相对稳定的，表中的记录是要长期保存的。

理解基本表的性质后，在设计数据库时，就能将基本表与中间表、临时表区分开来。

4. 范式标准

基本表及其字段之间的关系，应尽量满足第三范式。但是，满足第三范式的数据库设计，往往不是最好的设计。为了提高数据库的运行效率，常常需要降低范式标准：适当增加冗余，达到以空间换时间的目的。

〖例 2〗：有一张存放商品的基本表，如表 1 所示。“金额”这个字段的存在，表明该表的设计不满足第三范式，因为“金额”可以由“单价”乘以“数量”得到，说明“金额”是冗余字段。但是，增加“金额”这个冗余字段，可以提高查询统计的速度，这就是以空间换时间的作法。

在 Rose 2002 中，规定列有两种类型：数据列和计算列。“金额”这样的列被称为“计算列”，而“单价”和“数量”这样的列被称为“数据列”。

表 1 商品表的表结构

商品名称 商品型号 单价 数量 金额

电视机 29 吋 2,500 40 100,000

1. 通俗地理解三个范式

通俗地理解三个范式，对于数据库设计大有好处。在数据库设计中，为了更

好地应用三个范式，就必须通俗地理解三个范式(通俗地理解是够用的理解，并不是最科学最准确的理解)：

第一范式：1NF 是对属性的原子性约束，要求属性具有原子性，不可再分解；

第二范式：2NF 是对记录的惟一性约束，要求记录有惟一标识，即实体的惟一性；

第三范式：3NF 是对字段冗余性的约束，即任何字段不能由其他字段派生出来，它要求字段没有冗余。

没有冗余的数据库设计可以做到。但是，没有冗余的数据库未必是最好的数据库，有时为了提高运行效率，就必须降低范式标准，适当保留冗余数据。具体做法是：在概念数据模型设计时遵守第三范式，降低范式标准的工作放到物理数据模型设计时考虑。降低范式就是增加字段，允许冗余。

6. 要善于识别与正确处理多对多的关系

若两个实体之间存在多对多的关系，则应消除这种关系。消除的办法是，在两者之间增加第三个实体。这样，原来一个多对多的关系，现在变为两个一对多的关系。要将原来两个实体的属性合理地分配到三个实体中去。这里的第三个实体，实质上是一个较复杂的关系，它对应一张基本表。一般来讲，数据库设计工具不能识别多对多的关系，但能处理多对多的关系。

〖例 3〗：在“图书馆信息系统”中，“图书”是一个实体，“读者”也是一个实体。这两个实体之间的关系，是一个典型的多对多关系：一本图书在不同时间可以被多个读者借阅，一个读者又可以借多本图书。为此，要在二者之间增加第三个实体，该实体取名为“借还书”，它的属性为：借还时间、借还标志(0 表示借书，1 表示还书)，另外，它还应该有两个外键(“图书”的主键，“读者”的主键)，使它能与“图书”和“读者”连接。

7. 主键 PK 的取值方法

PK 是供程序员使用的表间连接工具，可以是一无物理意义的数字串，由程序自动加 1 来实现。也可以是有物理意义的字段名或字段名的组合。不过前者比后者好。当 PK 是字段名的组合时，建议字段的个数不要太多，多了不但索引占用空间大，而且速度也慢。

8. 正确认识数据冗余

主键与外键在多表中的重复出现，不属于数据冗余，这个概念必须清楚，事实上有许多人还不清楚。非键字段的重复出现，才是数据冗余！而且是一种低级冗余，即重复性的冗余。高级冗余不是字段的重复出现，而是字段的派生出现。

〖例 4〗：商品中的“单价、数量、金额”三个字段，“金额”就是由“单价”乘以“数量”派生出来的，它就是冗余，而且是一种高级冗余。冗余的目的是为了提高处理速度。只有低级冗余才会增加数据的不一致性，因为同一数据，可能从不同时间、地点、角色上多次录入。因此，我们提倡高级冗余(派生性冗余)，反对低级冗余(重复性冗余)。

9. E--R 图没有标准答案

信息系统的 E--R 图没有标准答案，因为它的设计与画法不是惟一的，只要它覆盖了系统需求的业务范围和功能内容，就是可行的。反之要修改 E--R 图。尽管它没有惟一的标准答案，并不意味着可以随意设计。好的 E—R 图的标准是：结构清晰、关联简洁、实体个数适中、属性分配合理、没有低级冗余。

10. 视图技术在数据库设计中很有用

与基本表、代码表、中间表不同，视图是一种虚表，它依赖数据源的实表而

存在。视图是供程序员使用数据库的一个窗口，是基表数据综合的一种形式，是数据处理的一种方法，是用户数据保密的一种手段。为了进行复杂处理、提高运算速度和节省存储空间，视图的定义深度一般不得超过三层。若三层视图仍不够用，则应在视图上定义临时表，在临时表上再定义视图。这样反复交迭定义，视图的深度就不受限制了。

对于某些与国家政治、经济、技术、军事和安全利益有关的信息系统，视图的作用更加重要。这些系统的基本表完成物理设计之后，立即在基本表上建立第一层视图，这层视图的个数和结构，与基本表的个数和结构是完全相同。并且规定，所有的程序员，一律只准在视图上操作。只有数据库管理员，带着多个人员共同掌握的“安全钥匙”，才能直接在基本表上操作。请读者想想：这是为什么？

11. 中间表、报表和临时表

中间表是存放统计数据的表，它是为数据仓库、输出报表或查询结果而设计的，有时它没有主键与外键(数据仓库除外)。临时表是程序员个人设计的，存放临时记录，为个人所用。基表和中间表由 DBA 维护，临时表由程序员自己用程序自动维护。

12. 完整性约束表现在三个方面

域的完整性：用 Check 来实现约束，在数据库设计工具中，对字段的取值范围进行定义时，有一个 Check 按钮，通过它定义字段的值城。

参照完整性：用 PK、FK、表级触发器来实现。

用户定义完整性：它是一些业务规则，用存储过程和触发器来实现。

13. 防止数据库设计打补丁的方法是“三少原则”

(1) 一个数据库中表的个数越少越好。只有表的个数少了，才能说明系统的 E--R 图少而精，去掉了重复的多余的实体，形成了对客观世界的高度抽象，进行了系统的数据集成，防止了打补丁式的设计；

(2) 一个表中组合主键的字段个数越少越好。因为主键的作用，一是建主键索引，二是做为子表的外键，所以组合主键的字段个数少了，不仅节省了运行时间，而且节省了索引存储空间；

(3) 一个表中的字段个数越少越好。只有字段的个数少了，才能说明在系统中不存在数据重复，且很少有数据冗余，更重要的是督促读者学会“列变行”，这样就防止了将子表中的字段拉入到主表中去，在主表中留下许多空余的字段。所谓“列变行”，就是将主表中的一部分内容拉出去，另外单独建一个子表。这个方法很简单，有的人就是不习惯、不采纳、不执行。

数据库设计的实用原则是：在数据冗余和处理速度之间找到合适的平衡点。“三少”是一个整体概念，综合观点，不能孤立某一个原则。该原则是相对的，不是绝对的。“三多”原则肯定是错误的。试想：若覆盖系统同样的功能，一百个实体(共一千个属性) 的 E--R 图，肯定比二百个实体(共二千个属性) 的 E--R 图，要好得多。

提倡“三少”原则，是叫读者学会利用数据库设计技术进行系统的数据集成。数据集成的步骤是将文件系统集成为应用数据库，将应用数据库集成为主题数据库，将主题数据库集成为全局综合数据库。集成的程度越高，数据共享性就越强，信息孤岛现象就越少，整个企业信息系统的全局 E—R 图中实体的个数、主键的个数、属性的个数就会越少。

提倡“三少”原则的目的，是防止读者利用打补丁技术，不断地对数据库进行增删改，使企业数据库变成了随意设计数据库表的“垃圾堆”，或数据库表的

“大杂院”，最后造成数据库中的基本表、代码表、中间表、临时表杂乱无章，不计其数，导致企事业单位的信息系统无法维护而瘫痪。

“三多”原则任何人都可以做到，该原则是“打补丁方法”设计数据库的歪理学说。“三少”原则是少而精的原则，它要求有较高的数据库设计技巧与艺术，不是任何人都能做到的，因为该原则是杜绝用“打补丁方法”设计数据库的理论依据。

14. 提高数据库运行效率的办法

在给定的系统硬件和系统软件条件下，提高数据库系统的运行效率的办法是：

(1) 在数据库物理设计时，降低范式，增加冗余，少用触发器，多用存储过程。

(2) 当计算非常复杂、而且记录条数非常巨大时(例如一千万条)，复杂计算要先在数据库外面，以文件系统方式用 C++语言计算处理完成之后，最后才入库追加到表中去。这是电信计费系统设计的经验。

(3) 发现某个表的记录太多，例如超过一千万条，则要对该表进行水平分割。水平分割的做法是，以该表主键 PK 的某个值为界线，将该表的记录水平分割为两个表。若发现某个表的字段太多，例如超过八十个，则垂直分割该表，将原来的一个表分解为两个表。

(4) 对数据库管理系统 DBMS 进行系统优化，即优化各种系统参数，如缓冲区个数。

(5) 在使用面向数据的 SQL 语言进行程序设计时，尽量采取优化算法。

总之，要提高数据库的运行效率，必须从数据库系统级优化、数据库设计级优化、程序实现级优化，这三个层次上同时下功夫。

上述十四个技巧，是许多人在大量的数据库分析与设计实践中，逐步总结出来的。对于这些经验的运用，读者不能生帮硬套，死记硬背，而要消化理解，实事求是，灵活掌握

关系数据库规范化理论

1.1 函数依赖

1.1.1 函数依赖的基本概念

定义 1.1：设 R(U)是属性集 U 上的关系模式。X,Y 是 U 的子集。若对于 R(U)的任意一个可能的关系 r，t1,t2 是 r 中的任意两个元组，如果由 t1[X]=t2[X]Tt1[Y]=t2[Y]，（即不可能存在两个元组在 X 上的属性值相等,而在 Y 上的属性值不等）则称 X 函数确定 Y 或 Y 函数依赖于 X，记作 X→Y。

1.1.2 一些术语和记号

设 R(U)是属性集 U 上的关系模式。X,Y 是 U 的子集。

X→Y，但 Y 不包含于 X 则称 X→Y 是非平凡的函数依赖。

若不特别声明,我们总是讨论非平凡的函数依赖。

X→Y，但 Y ⊆ X 则称 X→Y 是平凡的函数依赖。

若 X→Y,则 X 叫做决定因子。

若 X→Y，Y→X，则记作 X←→Y。

若 Y 不函数依赖于 X,则记作 X ↛ Y。

例：判断以下函数依赖的对错

- sno→sname, cno→cname,(sno,cno)→grade
- sname→sno, Sno→cno, sno→Cname

补充：属性间的联系决定函数依赖关系

设 X、Y 均是 U 的子集

- X 和 Y 间联系是 1:1，则 X→Y,Y→X。
- X 和 Y 间联系是 M:1， 则 X→Y。
- X 和 Y 间联系是 M:N，则 X、Y 间不存在函数依赖。

例：STUDENTS(SNO,SNAME,SSEX,SAGE,SDEPT,SPLACE)

X Y

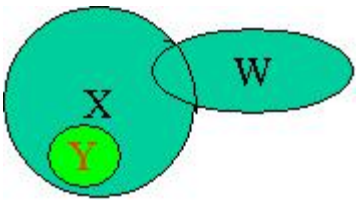
SNO → (SNAME,SSEX,SAGE)

SNO → SPLACE

SPLACE → SDEPT

SSEX → SDEPT

例：设关系 X,Y,W 为关系 R 中的三个属性组，属性关系如下图所示，问 X→Y, X→W, W→Y



各属上述何种函数依赖:

X→Y 为平凡函数依赖

X→W, W→Y 为非平凡函数依赖

补充：

定义 1.2：在 R(U)中,如果 X→Y，并且对于 X 的任何一个真子集 X' ,都有 X' $\not\rightarrow$ Y,则称 Y 对 X 完全函数依赖,记作:X→Y 。

若 X→Y,但 Y 不完全函数依赖于 X,则称 Y 对 X 部分函数依赖,记作 X $\xrightarrow{p}$ Y。

(一般，1:1 为完全函数依赖，m:1 为部分函数依赖)

定义 1.3：在 R(U)中,如果 X→Y,(Y?X)，Y $\not\rightarrow$ X,Y→Z,则称 Z 对 X 传递函数依赖。

1.2 关系规范化

定义 1.4 设 K 为 R〈U，F〉中的属性或属性组合，若 K → U 则 K 为 R 的候选码。

主码：若候选码多于一个，则选定其中的一个为主码（Primary key）

主属性：包含在任何一个候选码中的属性,叫做主属性(Prime attribute)

非主属性：不包含在任何码中的属性称为非主属性(Nonprime attribute)

最简单的情况：单个属性是码。

最极端的情况：整个属性组是码,称为全码(All-key)

例：关系模式 R(P,W,A),属性 P 表示演奏者,W 表示作品,A 表示听众。假设一个演奏者可以演奏多个作品,某一作品可被多个演奏者演奏。听众也可以欣赏不同演奏者的不同作品,这个关系模式的码为(P,W,A),即 All-key。

定义 1.5 关系模式 R 中属性或属性组 X 并非 R 的码,但 X 是另一个关系模式的码,则称 X 是 R 的外部码(Foreign key)也称外码。

主码与外部码提供了一个表示关系间联系的手段。

范式

关系模式满足的确定约束条件称为范式，根据满足约束条件的级别不同，范式由低到高分为 1NF,2NF,3NF,BCNF,4NF,5NF 等。不同的级别范式性质不同。满足最低要求的叫第一范式,简称 INF。在第一范式中满足进一步要求的为第二范式,其余以此类推。

R 为第几范式就可以写成 R∈xNF。

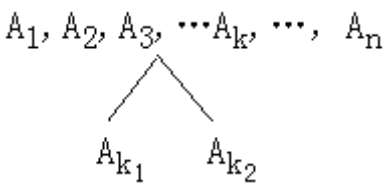
对于各种范式之间的联系有 5NF⊂4NF⊂ BCNF⊂ 3NF⊂ 2NF⊂ INF 成立。

一个低一级范式的关系模式，通过模式分解可以转换为若干个高一级范式的关系模式的集合,这种过程就叫规范化。

一、第一范式(1NF)

关系模式的的每一个属性都是不可再分的,则该关系模式称为第一范式。

例 1：



例 2：工资(工号，姓名，工资(基本工资，年绩津贴，煤电补贴))

△ 不满足 1NF 的关系称为非规范化关系。

△ 关系数据模型不能存储上两个例子（非规范化关系）

在关系数据库中不允许非规范化关系的存在。

二、第二范式

若 R∈INF,且每一个非主属性完全函数依赖于码,则 R∈2NF。

例：关系模式 S-L-C(SNO，SDEPT，SLOC，CNO，G) 中 SLOC 为学生的住处,并且每个系的学生住在同一个地方。

这里主码为(SNO,CNO)。函数依赖有：

(SNO,CNO) →G)

$$SNO \rightarrow SDEPT(SNO,CNO) \xrightarrow{p} SDEPT$$

$$SNO \rightarrow SLOC(SNO,CNO) \xrightarrow{p} SLOC,$$

一个关系模式 R 不属于 2NF,就会产生插入异常(如没有选课的学生记录插不进去)、删除异常（删除选课记录会将学生信息删除）、冗余度大（如系、地址都重复存放）。

分析上面的例子,可以发现问题在于有两种非主属性。一种如 G,它对码是完全函数依赖。另一种如 SDEPT、SLOC 对码不是完全函数依赖。解决的办法是用投影分解把关系模式 S-L-C 分解为两个关系模式。

SC(SNO,CNO,G)

S-L(SNO,SDEPT,SLOC)

关系模式 SC 的码为(SNO,CNO)，关系模式 S-L 的码为 SNO，这样就使得非主属性对码都是完全函数依赖

三、第三范式

关系模式 R〈U,F〉中若不存在这样的码 X,属性组 Y 及非主属性 Z(Z Y)使得 X→Y,（Y X）Y→Z,成立,即如果 R 的任何一个非主属性都不传递依赖于它的任何一个侯选关键字，则称 R〈U，F〉∈3NF。

可以证明,若 R∈3NF，则每一个非主属性既不部分依赖于码也不传递依赖于码。

在关系模式 SC 没有传递依赖，关系模式 S-L 存在非主属性对码传递依赖。在 S-L 中,由 SNO→SDEPT,（SDEPT

SNO）,SDEPT→SLOC,可得 SNO SLOC。因此 SC∈3NF,而 S-L 3NF。

一个关系模式 R 若不是 3NF,就会产生插入异常、删除异常、冗余度大等问题。

解决的办法同样是将 S-L 分解为：

S-D(SNO,SDEPT)

D-L(SDEPT,SLOC)

分解后的关系模式 S-D 与 D-L 中不再存在传递依赖。

四、BC 范式

关系模式 R〈U,F〉∈1NF。若 X→Y 且 Y?X 时 X 必含有码,则 R〈U,F> ∈BCNF。也就是说,关系模式 R〈U,F〉中,若每一个决定因素都包含码,则 R〈U,F> ∈BCNF。

由 BCNF 的定义可以得到以下结论:

下面用几个例子说明属于 3NF 的关系模式有的属于 BCNF,但有的不属于 BCNF。

例 1： 关系模式 SJP(S,J,P)中,S 是学生,J 表示课程,P 表示名次。每一个学生选修每门课程的成绩有一定的名次,每门课程中每一名次只有一个学生（即没有并列名次）。由语义可得到下面的函数依赖：

(S,J)→P ，(J,P)→S

所以(S,J)与(J,P)都可以作为候选码。这两个码各由两个属性组成,而且它们是相交的。这个关系模式中显然没有属性对码传递依赖或部分依赖。所以 SJP∈3NF,而且除(S,J)与(J,P)以外没有其它决定因素,所以 SJP∈BCNF。

例 2： 关系模式 STJ(S,T,J)中,S 表示学生,T 表示教师,J 表示课程。每一教师只教一门课。每门课有若干教师,某一学生选定某门课,就对应一个固定的教师。由语义可得到如下的函数依赖。

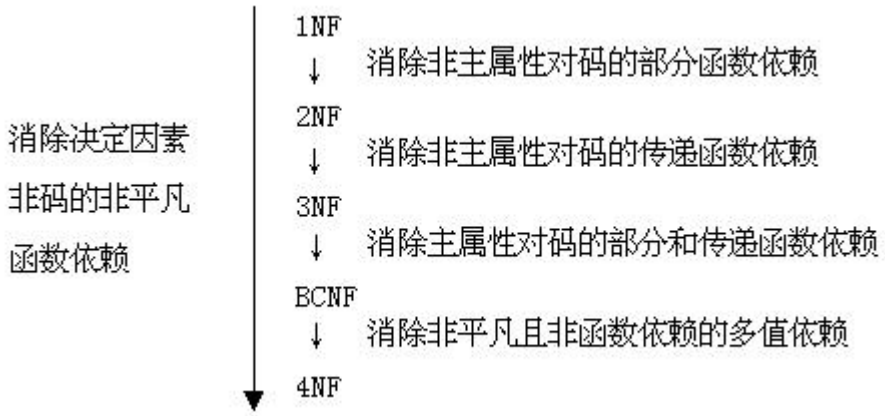
（S,J）→T , (S,T) →J

是第三范式，但是存在 T→J，而 T 不是码，所以不是了 BC 范式。

第三节 关系模式的分解准则

1．3 关系模式的分解准则

关系模式的规范化过程是通过对关系模式的分解来实现的。把低一级的关系模式分解为若干个高一级的关系模式。这种分解不是唯一的。



一个低级范式的关系模式，通过分解（投影）方法可转换成多个高一级范式的关系模式的集合，这种过程称为规范化。

规范化的方式是进行模式分解，模式分解的原则是与原模式等价，模式分解的标准是：

- 模式分解具有无损连接性
- 模式分解能够保持函数依赖

见 P70 页表格

举例：关系规范化过程

第一范式（1NF）：如果一关系模式，它的每一个分量是不可分的数据项，即其域为简单域，则此关系模式为第一范式。

例：将学生简历及选课等数据设计成一个关系模式 STUDENT， 其表示为：

STUDENT（SNO,SNAME,AGE,SEX,CLASS,DEPTNO,DEPTNAME,CNO,CNAME,SCORE,CREDIT）

设该关系模式满足下列函数依赖：

F={SNO-->SNAME, SNO-->AGE, SNO-->SEX,
SNO-->CLASS,CLASS-->DEPTNO, DEPTNO-->DEPTNAME,

CNO-->CNAME,SNO.CNO-->SCORE, CNO-->CREDIT}

由于该关系模式的每一属性对应的域为简单域，即其域值不可再分，符合第一范式定义，所以 STUDENT 关系模式为第一范式。

第二范式（2NF）：若关系模式 R?1NF，且每个非主属性完全函数依赖于码，则称 R?2NF。

分析一下关系模式 STUDENT， 它是不是 2NF ？

属性组（SNO，CNO）为关系 STUDENT 的码。

例如：SNAME 非主属性，根据码的特性具有：SNO.CNO??SNAME

根据 STUDENT 关系模式已知函数依赖集，下列函数依赖成立：SNO??SNAME

所以 SNO.CNO??SNAME, SNAME 对码是部分函数依赖。同样方法可得到除 SCORE 属性外，其它非主属性对码也都是部分函数依赖。所以 STUDENT 关系模式不是 2NF。

当关系模式 R 是 1NF 而不是 2NF 的模式时，对应的关系有何问题呢？我们分析 STUDENT 关系模式，会有下列问题：

- 存在大量的冗余数据：当一个学生在学习多门课程后，他的人事信息重复出现多次。
- 根据关系模型完整性规则，主码属性值不能取空值。那么新生刚入学，还未选修课程时，该元组就不能插入该关系中。这种情况称为插入异常。
- 同样还有删除异常，则会丢失信息

解决上述问题方法是将大的模式分解成多个小的模式，分解后的模式可满足更高级的范式的要求。

第三节 关系模式的分解准则

例如：将 STUDENT 中对码完全依赖的属性和部分函数依赖的属性分别组成关系。即将 STUDENT 关系模式分解成三个关系模式：

STUDENT1 （SNO，SNAME，AGE，SEX，CLASS，DEPTNO，DEPTNAME）

COURSE（CNO，CNAME，CREDIT）

SC（SNO，CNO，SCORE）

在分解后的每一个关系模式中，非主属性对码是完全函数依赖，所以上述三个关系模式均为 2NF。

第三范式（3NF）:若关系模式 R(U, F)为第一范式，不存在非主属性对码的传递依赖，则称 R(U, F)为 3NF。其中 U 为关系模式的属性全集，F 为关系模式所满足的函数依赖集。

分析关系模式 STUDENT1，存在着下列函数依赖：SNO-->CLASS，CLASS-->SNO，CLASS-->DEPTNO。所以关系模式 STUDENT1 属性间存在传递依赖，它不是 3NF。

如果关系模式 R 为 2NF 而不是 3NF。即存在数据冗余，插入和删除会出现异常。

- 例如在 STUDENT1 关系中，对每一学生其 DEPTNO，DEPTNAME 将重复出现。
- 当新成立一个系后，在尚未有学生时，该系的信息插入不到该关系中

要消除上述问题，必须对模式分解，消除传递依赖。将 STUDENT1 分解为下列模式：

STUDENT2（SNO，SNAME，AGE，SEX，CLASS）

CLASS1（CLASS，DEPTNO，DEPTNAME）

分解后的关系模式 STUDENT2 不再存在传递依赖，所以它是 3NF。但 CLASS1 关系模式虽只有三个属性，但还存在传递依赖。

CLASS-->DEPTNO，DEPTNO-->CLASS， DEPTNO-->DEPTNAME

DEPTNAME 对 CLASS 为传递依赖。所以对 CLASS1 还要进行分解，分解为下列模式：

CLASS2（CLASS，DEPTNO）

DEPARTMENT（DEPTNO，DEPTNAME）

关系模式 STUDENT 经过上述分解处理，分解成下列关系模式：

STUDENT2（SNO，SNAME，AGE，SEX，CLASS）

CLASS2（CLASS，DEPTNO）

DEPARTMENT（DEPTNO，DEPTNAME）

COURSE（CNO，CNAME，CREDIT）

SC（SNO，CNO，SCORE）

BCNF 范式：BCNF 是修正的第三范式。

设关系模式 R(U，F)?1NF，若 X?Y，而 Y 不包含在 X 中，那么 X 必含有码，则 R(U, F)为 BCNF。换句话说，每个决定因素都包含有码，则关系模式为 BCNF。

3NF 关系模式和 BCNF 的关系模式之间有下列关系：

如果 R∈ BCNF，则 R∈3NF，反之不成立。

BCNF 的定义消除了 3NF 模式中可能存在主属性对码的部分函数依赖和传递函数依赖。