关系数据库设计

[如何理解关系型数据库的常见设计范式？ - 刘慰的回答 - 知乎]

[数据库范式之第二范式]

范式

范式是"符合某一种级别的关系模式的集合，表示一个关系内部各属性之间的联系的合理化程度"。

实际上你可以把它粗略地理解为**一张数据表的表结构所符合的某种设计标准的级别。**就像家里装修买建材，最环保的是 E0 级，其次是 E1 级，还有 E2 级等等。数据库范式也分为1NF，2NF，3NF，BCNF，4NF，5NF。一般在我们设计关系型数据库的时候，最多考虑到 BCNF 就够。符合高一级范式的设计，必定符合低一级范式，例如符合2NF 的关系模式，必定符合 1NF 。

> 为什么最多考虑到BCNF，参见反范式化

第一范式（1NF）

概念

所谓第一范式（1NF）是指在关系模型中，对域添加的一个规范要求，所有的域都应该是原子性的，即数据库表的每一列都是不可分割的原子数据项，而不能是集合，数组，记录等非原子数据项。即实体中的某个属性有多个值时，必须拆分为不同的属性。在符合第一范式（1NF）表中的每个域值只能是实体的一个属性或一个属性的一部分。简而言之，**第一范式就是无重复的域。符合1NF的关系中的每个属性都不可再分。**

如下表，就不符合1NF。

编号	品名	进货		销售		备注
		数量	单价	数量	单价	

实际上，**1NF是所有关系型数据库的最基本要求**，你在关系型数据库管理系统（RDBMS），例如SQL Server，Oracle，MySQL中创建数据表的时候，如果数据表的设计不符合这个最基本的要求，那么操作一定是不能成功的。也就是说，只要在RDBMS中已经存在的数据表，一定是符合1NF的。如果我们要在RDBMS中表现表中的数据，就得设计为下表的形式：

编号	品名	进货数量	进货单价	销售数量	销售单价	备注

通常上只要知道 1NF 的上述特点即可。但是为了严谨，这里举出三点不符合第一范式的例子。

反例

> 不符合第一范式的情况

重复组(单一字段中有多个有意义的值)

重复组通常会出现在会计账上，每一笔记录可能有不定个数的值。举例来说：

顾客	日期	数量
Pater	Monday	19.00,-28.20
Pater	Wednesday	-84.00
Sarah	Friday	100.00,150.00

'数量' 就是所谓的重复组了，而在这种情况下这份数据就不符合第一范式。想要消除重复组的话，只要把每笔记录都转化为单一记录即可：

顾客	日期	数量
Pater	Monday	19.00
Pater	Monday	-28.20
Pater	Wednesday	-84.00
Sarah	Friday	100.00
Sarah	Friday	150.00

缺少唯一识别码

一样是在交易这个例子中，同一天同一个人买了同样的数量，这样的交易做了两次：

顾客	日期	数量
Pater	Monday	19.00
Pater	Monday	19.00

如上所示，这两笔交易可以说是一模一样，也就是说如果只靠这些数据我们没有办法分辨这两笔记录。我们之所以说它不符合第一范式，是因为上面这样的表示法欠缺一个唯一识别码，可以是一个字段，也可以是一组字段，而且可以保证在这个数据中唯一识别码不会重复出现。要将它正规化到符合第一范式的原则只需要加入一个唯一识别码即可：

交易ID	顾客	日期	数量
1	Pater	Monday	19.00
2	Pater	Monday	19.00

用很多字段来表达同一个事实

在同一个数据表里用多个字段来表达同一个事情也是违反第一范式的：

交易ID	人物	不喜欢的食物(1)	不喜欢的食物(2)	不喜欢的食物(3)
1	Pater	梨子	苹果	
2	Jim	西瓜		梨子
3	Airing	西瓜	苦瓜	梨子
4	Tom		老鼠	葡萄

懂得数据库基础的人一看就知道这是一个很糟糕的设计，甚至连觉得其都称不上是一个数据表。没错，其实1NF是很简单，它是所有关系型数据库设计必须要满足的基础。

1NF进行了哪些改进即可。其改进是，**2NF在1NF的基础之上，消除了非主属性对于码的部分函数依赖。**

异常

如果仅仅符合1NF的设计，仍然会存在**数据冗余过大，插入异常，删除异常，修改异常**的问题。

例如下面学生表的设计：

Sno	Sname	Sdept	Mname（系主任）	Cname	Grade
1022211101	李小明	经济系	王强	高数	95
1022211101	李小明	经济系	王强	英语	87
1022211101	李小明	经济系	王强	化学	76
1022211102	张莉莉	经济系	王强	高数	72
1022211102	张莉莉	经济系	王强	英语	98
1022211102	张莉莉	经济系	王强	操作系统	88
1022511101	高芳芳	法律系	刘玲	高数	82
1022511101	高芳芳	法律系	刘玲	法学	82

表1 学生成绩表

- **数据冗余过大**：每一名学生的学号、姓名、系名等数据重复多次。每个系与对应的系主任的数据也重复多次。
- **插入异常**：假如学校新建了一个系，但是暂时还没有招收任何学生（比如3月份就新建了，但要等到8月份才招生），那么是无法将系名与系主任的数据单独地添加到数据表中去的 （注）

  > 注：根据三种关系完整性约束中实体完整性的要求，关系中的码所包含的任意一个属性都不能为空，所有属性的组合也不能重复。为了满足此要求，图中的表，只能将学号与课名的组合作为码，否则就无法唯一地区分每一条记录。

- **删除异常**：假如将某个系中所有学生相关的记录都删除，那么所有系与系主任的数据也就随之消失。

  > 一个系所有学生都没有了，并不表示这个系就没有了。

- **修改异常**：若李小明转系到法律系，则为保证数据的一致性，需要修改三条记录中系与系主任的数据。

正因为仅符合1NF的数据库设计存在着这样那样的问题，我们需要提高设计标准，去掉导致上述四种问题的因素，使其符合更高一级的范式（2NF），这就是所谓的“规范化”。

函数依赖

概念

某个属性集决定另一个属性集时，称另一属性集依赖于该属性集。

设 $R(U)$ 是一个属性集 U 上的关系模式， X 和 Y 是 U 的子集。

若对于 $R(U)$ 的两个可能的关系 $r1$、$r2$，若 $r1[x]=r2[x]$，则 $r1[y]=r2[y]$，称 X 决定 Y，或者 Y 依赖X。

可以这么理解（并不是特别严格的定义）：**若在一张表中，在属性（或属性组）X 的值确定的情况下，必定能确定属性 Y 的值，那么就可以说 Y 函数依赖于 X，写作 $X \to Y$ 。**

> $X \to Y$，在数据表中，不存在任意两条记录，它们在 X 属性（或属性组）上的值相同，而在 Y 属性上的值不同。可能存在两条记录，它们在 Y 属性（或属性组）上的值相同，而在 X 属性上的值不同。
>
> 学号 $\to$ 姓名，学号相同，姓名肯定相同。姓名相同，学号不一定相同。

记 $X \to Y$ 表示 X 函数决定 Y，也可以说 Y 函数依赖于 X。

引申概念

完全函数依赖

在一张表中，若 $X \to Y$，且对于 X 的任何一个真子集（假如属性组 X 包含超过一个属性的话），$X' \to Y$ 不成立，那么我们称 Y 对于 X **完全函数依赖**，记作如下 $X \xrightarrow{F} Y$

- $学号 \xrightarrow{F} 姓名$：姓名完全依赖于学号
- $(学号，课名) \xrightarrow{F} 分数$：分数完全依赖于（学号，课名）

  > （同一个的学号有多门课的分数，同一个课名对应的分数也不确定）

部分函数依赖

假如 Y 函数依赖于 X，但同时 Y 并不完全函数依赖于 X，那么我们就称 Y 部分函数依赖于 X，记作 $X \xrightarrow{P} Y$

- $(学号，课名) \xrightarrow{P} 姓名$：姓名部分依赖于（学号，课名）

  > 只需要一个学号就够了。
- 部分函数依赖，只依赖 X 的一部分就够了

传递函数依赖

如果 $X \to Y$，$Y \to Z$，Y 不是 X 的子集，Y 不函数决定 X，则称 Z 对 X 传递函数依赖。记作 $X \xrightarrow{T} Y$

码

设 K 为某表中的一个属性或属性组，若除 K 之外的所有属性都**完全**函数依赖于 K（这个"完全"不要漏了），那么我们称 K 为**候选码**，亦称为**码。**在实际中我们通常可以理解为：**假如当 K 确定的情况下，该表除 K 之外的所有属性的值也就随之确定，那么 K 就是码。**一张表中可以有超过一个码。（实际应用中为了方便，通常选择其中的一个码作为**主码**、**键码**、**主键**）

> 假如 A 是码，那么所有包含了 A 的属性组，如 $(A，B), (A, C), (A，B，C)$等等，都不是码了。
>
> 因为作为码的要求里有一个"**完全**函数依赖"。

问：关于码的定义，如果除K之外的所有属性都完全函数依赖于K时才能称K为码，那么在判断2NF时又怎么会存在非主属性对码的部分函数依赖这种情况?

答：在"码"的定义中，除 K 之外的所有属性应该看成是一个集合 U（也就是一个整体），也就是说，只有 K 能够完全函数决定 U 中的每一个属性，那么 K 才是码。如果 K 只是能够完全函数决定 U 中的一部分属性，而不能完全函数决定另外一部分属性，那么 K 不是码。**非主属性看成整体，唯一确定一条记录的就是码。**

> 比如有关系模式 R (Sno, Sname, Cno, Cname, Sdept, Sloc, Grade)，其中函数依赖集为 F= { Sno → Sname, Sno → Sdept, Sdept → Sloc，Sno → Sloc, Cno → Cname, (Sno, Cno) → Grade }
>
> 那么 R 中的码只能是 (Sno, Cno)，Sno 或 Cno 并不能完全函数决定除 Sno / Cno 之外的所有其他属性（其实就是不能决定 Grade)，所以单独的 Sno 与 Cno 并不能作为码。
>
> 所以可得到主属性：Sno, Cno
> 非主属性：Sname, Cname, Sdept, Sloc, Grade
>
> R 中存在非主属性 Cname 对于码 (Sno, Cno) 的部分函数依赖 (Cno → Cname) 。（还有很多别的例子就不一一列举了）。所以 R 不符合 2NF 的要求。

- **超码（ super key ）**是一个或多个属性的集合，这些属性上的取值保证可以唯一识别出关系中的元组 。
  - 候选码是一个最小的超码，即它是一组构成超码的属性集 . 但这组属性的任意子集都不是超码。
  - 关系的一个候选码被选作主码 （ prime key ） 。
- **外码 （ foreign key ）**：对于参照关系中的每个元组来说，它在外码属性上的取值肯定等于被参照关系中某个元组在主码上的取值。
- **主属性**：在一个关系中,如一个属性是构成某一个候选关键字的属性集中的一个属性,则称它为主属性。
- **非主属性**：不包含在任何一个候选码中的属性称为非主属性。

平凡函数依赖

当关系中属性集合 Y 是属性集合 X 的子集即 $Y \subseteq X$ 时，那必然存在存在函数依赖 $X \to Y$，即一组属性函数决定它的所有子集（整体决定部分），这种函数依赖在所有关系中都满足，所以称为平凡函数依赖。

非平凡函数依赖

若关系中 $Y \nsubseteq X$ 且 $X \to Y$，这种函数依赖并不再所有关系中成立，所以称为非平凡函数依赖。

函数依赖和属性的关系

属性之间有三种关系，但并不是每一种关系都存在函数依赖。

设 $R(U)$ 是属性集 U 上的关系模式，X、Y 是 U 的子集：

- 如果 X 和 Y 之间是 一对一关系，如学校和校长之间就是 $1:1$ 关系，则 $X \to Y$ 且 $Y \to X$。
- 如果 X 和 Y 之间是 一对多关系，如年龄和姓名之间就是 $1:n$ 关系，则 $Y \to X$。
- 如果 X 和 Y 之间是 多对多关系，如学生和课程之间就是 $m:n$ 关系，则 X 和 Y 之间不存在函数依赖。

第二范式（2NF）

概念

2NF在1NF的基础之上，消除了非主属性对于码的部分函数依赖。

第二范式的规则是要求数据表里的所有数据都要和该数据表的主键有完全依赖关系；如果有哪些数据只和主键的一部分有关的话，就得把它们独立出来变成另一个数据表。如果一个数据表的主键只有单一一个字段的话，它就一定符合第二范式。

下表不符合2NF，为了让表符合2NF的要求，需要消除这些部分函数依赖。唯一的办法是将大数据表拆分成两个或者更多个更小的数据表，在拆分的过程中，要达到更高一级范式的要求，这个过程叫做"**模式分解**"。

分解前

Sno	Sname	Sdept	Mname（系主任）	Cname	Grade
1022211101	李小明	经济系	王强	高数	95
1022211101	李小明	经济系	王强	英语	87
1022211101	李小明	经济系	王强	化学	76
1022211102	张莉莉	经济系	王强	高数	72
1022211102	张莉莉	经济系	王强	英语	98
1022211102	张莉莉	经济系	王强	操作系统	88
1022511101	高芳芳	法律系	刘玲	高数	82
1022511101	高芳芳	法律系	刘玲	法学	82

表2 学生成绩表

$\{Sno, Cname\}$ 为主键，$Sname$、$Sdept$、$Mname$、$Grade$为非主属性，有如下函数依赖：

- $\{Sno, Cname\} \xrightarrow{F} Grade$

  > Grade 完全函数依赖于主键，它没有任何冗余数据，每个学生的每门课都有特定的成绩。

- $Sno \xrightarrow{F} Sdept \Rightarrow \{Sno, Cname\} \xrightarrow{P} Sdept$
- $Sno \xrightarrow{F} Sname \Rightarrow \{Sno, Cname\} \xrightarrow{P} Sname$
- $Sdept \xrightarrow{F} Mname \Rightarrow \{Sno, Cname\} \xrightarrow{P} Mname$

> Sname, Sdept 和 Mname 都部分依赖于主键，当一个学生选修了多门课时，这些数据就会出现多次，造成大量冗余数据。

检查数据表里的每个字段，确认它们是不是都完全依赖于主键， 这样才能知道这个数据表是不是符合第二范式； 如果不是的话，就把那些不完全相关的字段移到独立的数据表里。

分解后

Sno	Cname	Grade
1022211101	高数	95
1022211101	英语	87
1022211101	化学	76
1022211102	高数	72
1022211102	英语	98
1022211102	操作系统	88
1022511101	高数	82
1022511101	法学	82

表3 学生选课表

$\{Sno, Cname\}$ 为主键，$Grade$为非主属性，有如下函数依赖：

- $\{Sno, Cname\} \xrightarrow{F} Grade$

> Grade 完全函数依赖于主键，符合 2NF

Sno	Sname	Sdept	Mname（系主任）
1022211101	李小明	经济系	王强
1022211102	张莉莉	经济系	王强
1022511101	高芳芳	法律系	刘玲

表4 学生信息表

Sno 为主键，$Sname$、$Sdept$、$Mname$为非主属性，有如下函数依赖：

- $Sno \xrightarrow{F} Sname$
- $Sno \xrightarrow{F} Sdept$
- $Sno \xrightarrow{F} Sdept$，$Sdept \xrightarrow{F} Mname => Sno \xrightarrow{T} Mname$

问题

现在我们来看一下，进行同样的操作，是否还存在着之前的那些问题？

- 李小明转系到法律系
  - 只需要修改一次李小明对应的系的值即可。——有改进
- 数据冗余是否减少了？

    - 学生的姓名、系名与系主任，不再像之前一样重复那么多次了。——有改进
- 删除某个系中所有的学生记录
    - 该系的信息仍然全部丢失。——无改进
- 插入一个尚无学生的新系的信息。
    - 因为学生表的码是学号，不能为空，所以此操作不被允许。——无改进

结论

所以说，仅仅符合2NF的要求，很多情况下还是不够的，而出现问题的原因，在于仍然存在非主属性**系主任**对于码**学号**的传递函数依赖。

第三范式（3NF）

概念

3NF在2NF的基础之上，消除了非主属性对于码的传递函数依赖。也就是说， 如果存在非主属性对于码的传递函数依赖，则不符合3NF的要求，要确保所有不是键的字段都和彼此没有相依关系。

在第三范式里，**所有的非键属性都必须和每个候选键有直接相关。**

分解后

- **表3学生选课表**：主码为（学号，课名），主属性为**学号**和**课名，**非主属性只有一个，为分数，分数完全依赖于学号和课名，符合3NF的要求。
- **表4学生信息表**，主码为**学号**，主属性为**学号**，非主属性为**姓名**、**系名**和**系主任。**因为系名完全依赖于学号，同时 系主任完全依赖于系名，所以存在非主属性**系主任**对于码**学号**的传递函数依赖，不符合3NF的要求。

对表4学生信息表进行模式分解

Sno	Sname	Sdept
1022211101	李小明	经济系
1022211102	张莉莉	经济系
1022511101	高芳芳	法律系

表5 学生信息表

Sdept	Mname（系主任）
经济系	王强
经济系	王强
法律系	刘玲

表6 系信息表

问题

现在我们来看一下，进行同样的操作，是否还存在着之前的那些问题?

- 删除某个系中所有的学生记录
  该系的信息不会丢失。——有改进
- 插入一个尚无学生的新系的信息。
  因为系表与学生表目前是独立的两张表，所以不影响。——有改进
- 数据冗余更加少了。——有改进

结论

由此可见，符合3NF要求的数据库设计，**基本**上解决了数据冗余过大，插入异常，修改异常，删除异常的问题。当然，在实际中，往往为了性能上或者应对扩展的需要，经常 做到2NF或者1NF，但是作为数据库设计人员，至少应该知道，3NF的要求是怎样的。

特殊例子

订单编号(Order Number)	客户名称 (Customer Name)	单价(Unit Price)	数量 (Quantity)	小计 (Total)
1	Airing	20.00	3	60.00
2	Bob	10.00	2	20.00

在本例中，非主键字段完全依赖于主键订单编号,也就是说唯一的订单编号能导出唯一非主键字段值，符合第二范式。第三范式要求非主键字段之间不能有依赖关系，显然本例中小计依赖于非主键字段单价和数量，不符合第三范式。小计不应该放在这个数据表里面，只要把单价乘上数量就可以得到小计了；如果想要符合第三范式的话，就把小计拿掉吧。

不过在做查询的时候，本来用:

```
SELECT Order.Total FROM Order
```

就要改成用：

```
SELECT UnitPrice * Quantity FROM Order
```

订单编号(Order Number)	客户名称 (Customer Name)	单价(Unit Price)	数量 (Quantity)
1	Airing	20.00	3
2	Bob	10.00	2

BCNF 范式

BCNF 就是在 3NF 的基础上消除**主属性**对于码的部分与传递函数依赖。

具有函数依赖集 U 的关系模式 R 属于 BCNF 的条件是， 对 U 中所有形如 $\alpha \rightarrow \beta$ 的函数依赖（其中 $\alpha \subseteq R$ 且 $\beta \subseteq R$），下面至少有一项成立:

- $\alpha \rightarrow \beta$ 是平凡的函数依赖 （即， $\beta \subseteq \alpha$） 。
- α 是模式 R 的一个超码。

若

1. 某公司有若干个仓库；
2. 每个仓库只能有一名管理员，一名管理员只能在一个仓库中工作；
3. 一个仓库可以存放多种物品，一种物品也可以存放在不同的仓库中。每种物品在每个仓库中都有对应的数量。

那么关系模式 仓库（仓库名，管理员，物品名，数量） 属于哪一级范式?

答：已知函数依赖集：仓库名 → 管理员，管理员 → 仓库名，（仓库名，物品名）→ 数量
码：（管理员，物品名），（仓库名，物品名）
主属性：仓库名、管理员、物品名
非主属性：数量
∵ 不存在非主属性对码的部分函数依赖和传递函数依赖。

∴ 此关系模式属于3NF。

基于此关系模式的关系（具体的数据）可能如表所示：

仓库名	管理员	物品名	数量
上海仓	张三	iPhone 5s	30
上海仓	张三	iPad Air	40
北京仓	李四	iPhone 5s	50
北京仓	李四	iPad Mini	60

既然此关系模式已经属于了 3NF，那么这个关系模式是否存在问题呢？我们来看以下几种操作：

- 先新增加一个仓库，但尚未存放任何物品，是否可以为该仓库指派管理员？——不可以，因为物品名也是主属性，根据实体完整性的要求，主属性不能为空。
- 某仓库被清空后，需要删除所有与这个仓库相关的物品存放记录，会带来什么问题？——仓库本身与管理员的信息也被随之删除了。
- 如果某仓库更换了管理员，会有什么问题？——这个仓库有几条物品存放记录，就要修改多少次管理员信息。

从这里我们可以得出结论，在某些特殊情况下，即使关系模式符合 3NF 的要求，仍然存在着插入异常，修改异常与删除异常的问题，仍然不是 "好" 的设计。

造成此问题的原因：存在着**主属性**对于码的部分函数依赖与传递函数依赖。（在此例中就是存在主属性【仓库名】对于码【（管理员，物品名）】的部分函数依赖。

采用 BCNF 范式解决此问题：

仓库（仓库名，管理员）
库存（仓库名，物品名，数量）

反范式化

> 问：范式的存在有什么好处?
>
> 范式可以避免数据冗余，减少数据库的空间，减轻维护数据完整性的麻烦。
>
> 范式以时间换空间

其实到了3NF，基本上就已经消除了**数据冗余**以及插入异常，删除异常，修改异常的问题。不过，如上例，订单表中，在执行SQL语句的时候就要多计算一步，或者上上例的学生表中，如果想得到学生的奖金就需要查两张表。如果在需要查询数据量特别大或者是经常性需要得到这些数据的时候，那么就可以适当的反范式化，稍微放松一点3NF的要求，达到以**空间换时间**的目的。

按照范式的规范设计出来的表，等级越高的范式设计出来的表越多。如第一范式可能设计出来的表可能只有一张表而已，再按照第二范式去设计这张表时就可能出来两张或更多张表，如果再按第三范式或更高的范式去设计这张表会出现更多比第二范式多的表。表的数量越多，当我们去查询一些数据，必然要去多表中去查询数据，这样查询的时间要比在一张表中查询中所用的时间要高很多。

也就是说我们所用的范式越高，对数据操作的性能越低。所以我们在利用范式设计表的时候，要根据具体的需求再去权衡是否使用更高范式去设计表。在一般的项目中，我们用的最多也就是第三范式，第三范式也就可以满足我们的项目需求，性能好而且方便管理数据。

当我们的业务所涉及的表非常多，经常会有多表发生关系，并且我们对表的操作要时间上要尽量的快，这时可以考虑我们使用“反范式”。反范式，故名思义，跟范式所要求的正好相反，在反范式的设计模式，我们可以允许适当的数据的冗余，用这个冗余去取操作数据时间的缩短。也就是**用空间来换取时间，把数据冗余在多个表中，当查询时可以减少或者是避免表之间的关联。**

在数据库中，读写效率大概是1：3到1：4之间，所以控制好反范式化的比例对优化数据库尤其重要。

当我们开始着手一个项目后，范式的应用是这样的变化的：

第三范式数据库的设计—–>当数据量越来越大，达到百万级时，经常要对一些多表数据进行大范围高频率进行操作———->范式数据库的设计————->网站的数据量再持续增长————->范式和反范式的数据库设计