21丨范式设计：数据表的范式有哪些，3NF指的是什么?

在日常工作中，我们都需要遵守一定的规范，比如签到打卡、审批流程等，这些规范虽然有一定的约束感，却是非常有必要的，这样可以保证正确性和严谨性，但有些情况下，约束反而会带来效率的下降，比如一个可以直接操作的任务，却需要通过重重审批才能执行。

实际上，数据表的设计和工作流程的设计很像，我们既需要规范性，也要考虑到执行时的方便性。

今天，我来讲解一下数据表的设计范式。范式是数据表设计的基本原则，又很容易被忽略。很多时候，当数据库运行了一段时间之后，我们才发现数据表设计得有问题。重新调整数据表的结构，就需要做数据迁移，还有可能影响程序的业务逻辑，以及网站正常的访问。所以在开始设置数据库的时候，我们就需要重视数据表的设计。

今天的课程你需要掌握以下几个方面的内容：

1. 数据库的设计范式都有哪些?
2. 数据表的键都有哪些?
3. 1NF、2NF 和 3NF 指的是什么?

数据库的设计范式都包括哪些

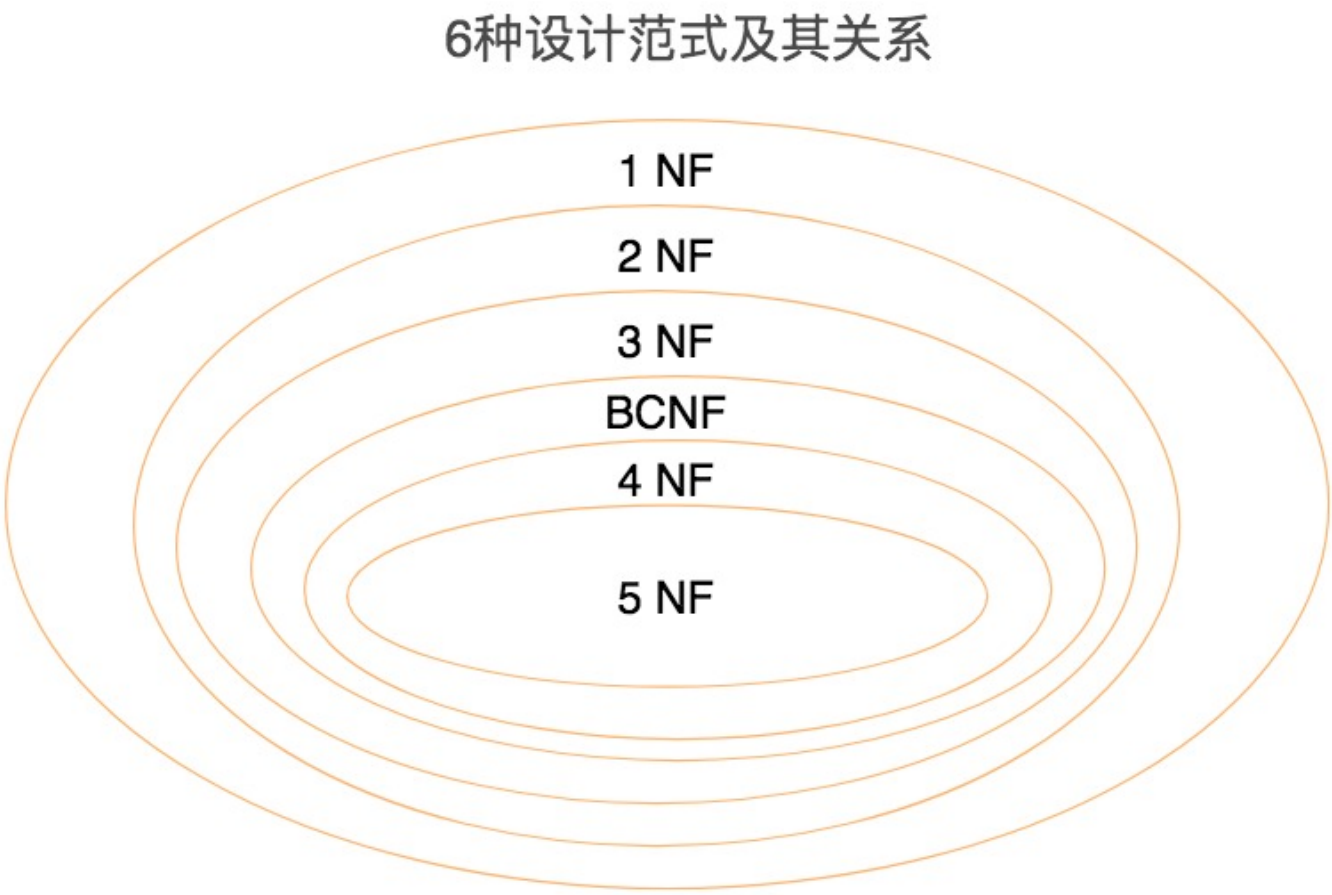
我们在设计关系型数据库模型的时候，需要对关系内部各个属性之间联系的合理化程度进行定义，这就有了不同等级的规范要求，这些规范要求被称为范式（NF）。你可以把范式理解为，一张数据表的设计结构需要满足的某种设计标准的级别。

目前关系型数据库一共有 6 种范式，按照范式级别，从低到高分别是：1NF（第一范式）、2NF（第二范式）、3NF（第三范式）、BCNF（巴斯 - 科德范式）、4NF（第四范式）和 5NF（第五范式，又叫做完美范式）。

数据库的范式设计越高阶，冗余度就越低，同时高阶的范式一定符合低阶范式的要求，比如满足 2NF 的一定满足 1NF，满足 3NF 的一定满足 2NF，依次类推。

你可能会问，这么多范式是不是都要掌握呢?

一般来说数据表的设计应尽量满足 3NF。但也不绝对，有时候为了提高某些查询性能，我们还需要破坏范式规则，也就是反规范化。



数据表中的那些键

范式的定义会使用到主键和候选键（因为主键和候选键可以唯一标识元组），数据库中的键（Key）由一个或者多个属性组成。我总结了下数据表中常用的几种键和属性的定义：

超键：能唯一标识元组的属性集叫做超键。

候选键：如果超键不包括多余的属性，那么这个超键就是候选键。

主键：用户可以从候选键中选择一个作为主键。

外键：如果数据表 R1 中的某属性集不是 R1 的主键，而是另一个数据表 R2 的主键，那么这个属性集就是数据表 R1 的外键。

主属性：包含在任一候选键中的属性称为主属性。

非主属性：与主属性相对，指的是不包含在任何一个候选键中的属性。

通常，我们也将候选键称之为“码”，把主键也称为“主码”。因为键可能是由多个属性组成的，针对单个属性，我们还可以用主属性和非主属性来进行区分。

看到上面的描述你可能还是有点懵，我举个简单的例子。

我们之前用过 NBA 的球员表（player）和球队表（team）。这里我可以把球员表定义为包含球员编号、姓名、身份证号、年龄和球队编号；球队表包含球队编号、主教练和球队所在地。

对于球员表来说，超键就是包括球员编号或者身份证号的任意组合，比如（球员编号）（球员编号，姓名）（身份证号，年龄）等。

候选键就是最小的超键，对于球员表来说，候选键就是（球员编号）或者（身份证号）。

主键是我们自己选定，也就是从候选键中选择一个，比如（球员编号）。

外键就是球员表中的球队编号。

在 player 表中，主属性是（球员编号）（身份证号），其他的属性（姓名）（年龄）（球队编号）都是非主属性。

从 1NF 到 3NF

了解了数据表中的 4 种键之后，我们再来看下 1NF、2NF 和 3NF，BCNF 我们放在后面讲。

1NF 指的是数据库表中的任何属性都是原子性的，不可再分。这很好理解，我们在设计某个字段的时候，对于字段 X 来说，就不能把字段 X 拆分成字段 X-1 和字段 X-2。事实上，任何的 DBMS 都会满足第一范式的要求，不会将字段进行拆分。

2NF 指的数据表里的非主属性都要和这个数据表的候选键有完全依赖关系。所谓完全依赖不同于部分依赖，也就是不能仅依赖候选键的一部分属性，而必须依赖全部属性。

这里我举一个没有满足 2NF 的例子，比如说我们设计一张球员比赛表 player_game，里面包含球员编号、姓名、年龄、比赛编号、比赛时间和比赛场地等属性，这里候选键和主键都为（球员编号，比赛编号），我们可以通过候选键来决定如下的关系：

(球员编号, 比赛编号) → (姓名, 年龄, 比赛时间, 比赛场地，得分)

上面这个关系说明球员编号和比赛编号的组合决定了球员的姓名、年龄、比赛时间、比赛地点和该比赛的得分数据。

但是这个数据表不满足第二范式，因为数据表中的字段之间还存在着如下的对应关系：

(球员编号) → (姓名，年龄)

(比赛编号) → (比赛时间, 比赛场地)

也就是说候选键中的某个字段决定了非主属性。你也可以理解为，对于非主属性来说，并非完全依赖候选键。这样会产生怎样的问题呢？

1. 数据冗余：如果一个球员可以参加 m 场比赛，那么球员的姓名和年龄就重复了 m-1 次。一个比赛也可能会有 n 个球员参加，比赛的时间和地点就重复了 n-1 次。
2. 插入异常：如果我们想要添加一场新的比赛，但是这时还没有确定参加的球员都有谁，那么就没法插入。
3. 删除异常：如果我要删除某个球员编号，如果没有单独保存比赛表的话，就会同时把比赛信息删除掉。
4. 更新异常：如果我们调整了某个比赛的时间，那么数据表中所有这个比赛的时间都需要进行调整，否则就会出现一场比赛时间不同的情况。

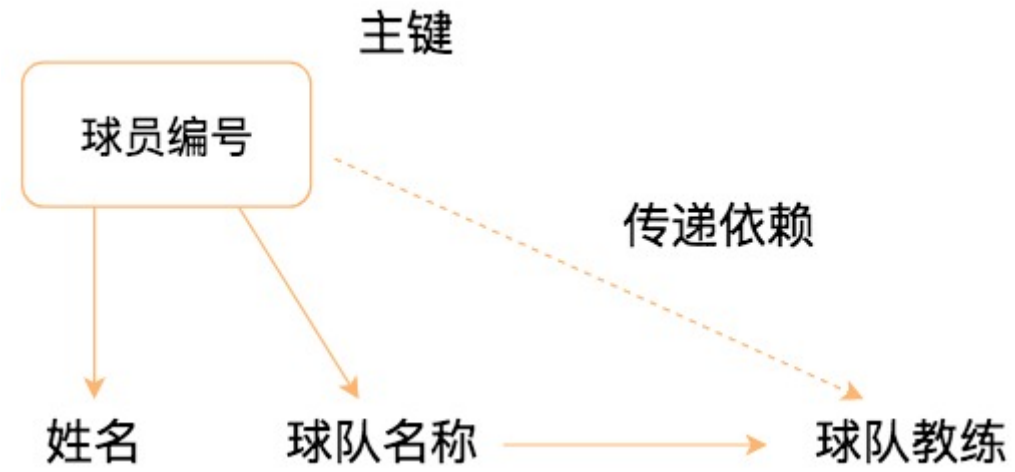
为了避免出现上述的情况，我们可以把球员比赛表设计为下面的三张表。

球员 player 表包含球员编号、姓名和年龄等属性；比赛 game 表包含比赛编号、比赛时间和比赛场地等属性；球员比赛关系 player_game 表包含球员编号、比赛编号和得分等属性。

这样的话，每张数据表都符合第二范式，也就避免了异常情况的发生。某种程度上 2NF 是对 1NF 原子性的升级。1NF 告诉我们字段属性需要是原子性的，而 2NF 告诉我们一张表就是一个独立的对象，也就是说一张表只表达一个意思。

3NF 在满足 2NF 的同时，对任何非主属性都不传递依赖于候选键。也就是说不能存在非主属性 A 依赖于非主属性 B，非主属性 B 依赖于候选键的情况。

我们用球员 player 表举例子，这张表包含的属性包括球员编号、姓名、球队名称和球队主教练。现在，我们把属性之间的依赖关系画出来，如下图所示：



你能看到球员编号决定了球队名称，同时球队名称决定了球队主教练，非主属性球队主教练就会传递依赖于球员编号，因此不符合 3NF 的要求。

如果要达到 3NF 的要求，需要把数据表拆成下面这样：

球员表的属性包括球员编号、姓名和球队名称；球队表的属性包括球队名称、球队主教练。

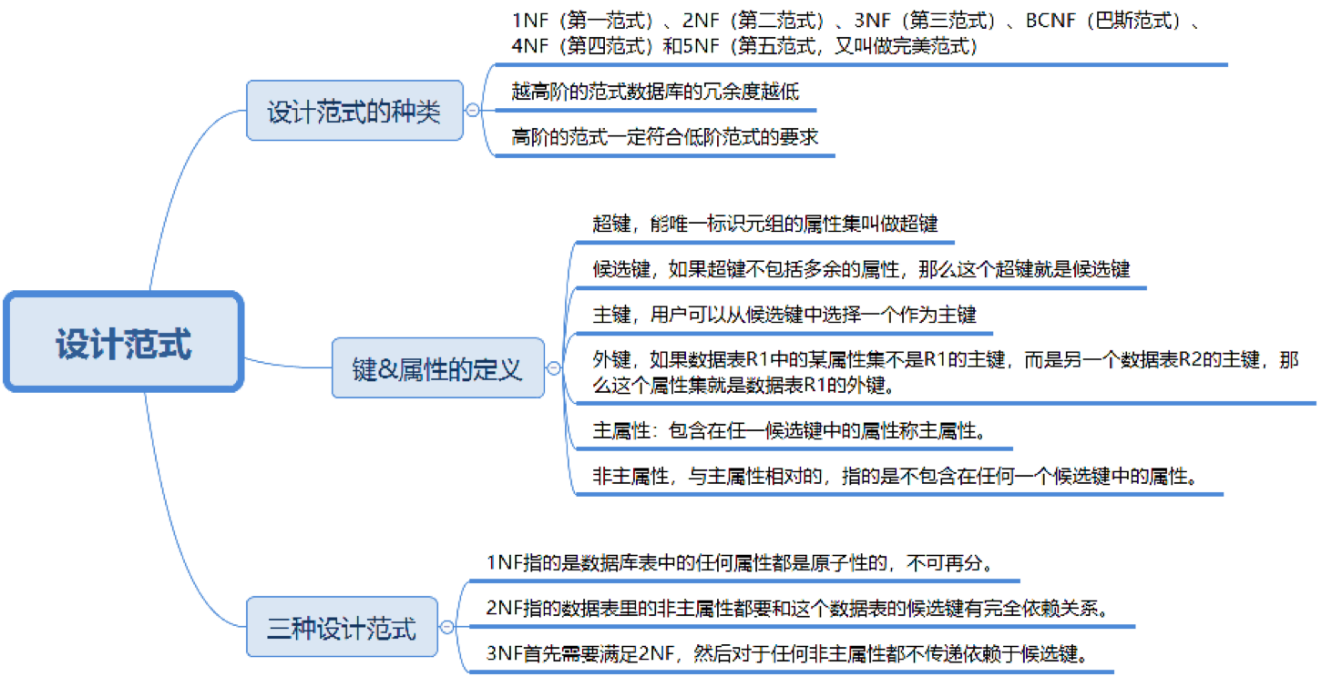
我再总结一下，1NF 需要保证表中每个属性都保持原子性；2NF 需要保证表中的非主属性与候选键完全依赖；3NF 需要保证表中的非主属性与候选键不存在传递依赖。

总结

我们今天讲解了数据表设计的三种范式。关系型数据库的设计都是基于关系模型的，在关系模型中存在着 4 种键，这些键的核心作用就是标识。

在这些概念的基础上，我又讲了 1NF，2NF 和 3NF。我们经常会与这三种范式打交道，利用它们建立冗余度小、结构合理的数据库。

有一点需要注意的是，这些范式只是提出了设计的标准，实际上设计数据表时，未必要符合这些原则。一方面是因为这些范式本身存在一些问题，可能会带来插入，更新，删除等的异常情况（这些会在下一讲举例说明），另一方面，它们也可能降低会查询的效率。这是为什么呢？因为范式等级越高，设计出来的数据表就越多，进行数据查询的时候就可能需要关联多张表，从而影响查询效率。



2NF 和 3NF 相对容易混淆，根据今天的内容，你能说下这两个范式之间的区别吗？另外，如果我们现在有一张学生选课表，包含的属性有学号、姓名、课程名称、分数、系别和系主任，如果要改成符合 3NF 要求的设计，需要怎么修改呢？