技术文章

[数据库三大范式](http://blog.csdn.net/u010425776/article/details/60968300)浅析

为了建立冗余较小、结构合理的数据库，设计数据库时必须遵循一定的规则。在关系型数据库中这种规则就称为范式。范式是符合某一种设计要求的总结。要想设计一个结构合理的关系型数据库，必须满足一定的范式。

真正要明白”范式（NF）”是什么意思，首先看下教材中的定义，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。

在实际开发中最为常见的设计范式有三个：

首先是第一范式（1NF）。  
符合1NF的关系（你可以理解为数据表。“关系”和“关系模式”的区别，类似于面向对象程序设计中”类“与”对象“的区别。”关系“是”关系模式“的一个实例，你可以把”关系”理解为一张带数据的表，而“关系模式”是这张数据表的表结构。1NF的定义为：符合1NF的关系中的每个属性都不可再分。表1所示的情况，就不符合1NF的要求。  


表1

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，就得设计为表2的形式：表2

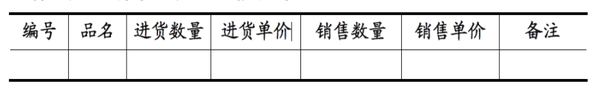


表2

但是仅仅符合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，例如对于表3中的设计：



表3

每一名学生的学号、姓名、系名、系主任这些数据重复多次。每个系与对应的系主任的数据也重复多次——数据冗余过大

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 ----—插入异常

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了（一个系所有学生都没有了，并不表示这个系就没有了）。**——删除异常**

假如李小明转系到法律系，那么为了保证数据库中数据的一致性，需要修改三条记录中系与系主任的数据。**——修改异常。**

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”。

第二范式

第二范式在第一范式的基础之上更进一层。是指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。

**函数依赖**：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 → Y。

表中的函数依赖关系例如：

系名 → 系主任

学号 → 系主任

（学号，课名） → 分数

但以下函数依赖关系则不成立：

学号 → 课名

学号 → 分数

课名 → 系主任

（学号，课名） → 姓名

**码**: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。码也可以理解为主键。

第二范式需要确保数据库表中的每一列都和主键相关，而不能只与主键的某一部分相关（主要针对联合主键而言）。也就是说在一个数据库表中，一个表中只能保存一种数据，不可以把多种数据保存在同一张数据库表中。

比如要设计一个订单信息表，因为订单中可能会有多种商品，所以要将订单编号和商品编号作为数据库表的联合主键，如下表所示。

订单信息表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 订单编号 | 商品编号 | 商品名称 | 数量 | 单位 | 价格 | 客户 | 所属单位 | 联系方式 |
| 001 | 1 | 挖掘机 | 1 | 台 | 1200000￥ | 张三 | 上海玖智 | 020-1234567 |
| 001 | 2 | 冲击钻 | 8 | 把 | 230￥ | 张三 | 上海玖智 | 020-1234567 |
| 002 | 3 | 铲车 | 2 | 辆 | 980000￥ | 李四 | 北京公司 | 010-1234567 |

这样就产生一个问题：这个表中是以订单编号和商品编号作为联合主键。这样在该表中商品名称、单位、商品价格等信息不与该表的主键相关，而仅仅是与商品编号相关。所以在这里违反了第二范式的设计原则。

而如果把这个订单信息表进行拆分，把商品信息分离到另一个表中，把订单项目表也分离到另一个表中，就非常完美了。如下所示。

订单信息表

|  |  |  |  |
| --- | --- | --- | --- |
| 订单编号 | 客户 | 所属单位 | 联系方式 |
| 001 | 张三 | 上海玖智 | 020-1234567 |
| 002 | 李四 | 北京公司 | 010-1234567 |

订单项目表

|  |  |  |
| --- | --- | --- |
| 订单编号 | 商品编号 | 数量 |
| 001 | 1 | 1 |
| 001 | 2 | 8 |
| 002 | 3 | 2 |

 商品信息表

|  |  |  |  |
| --- | --- | --- | --- |
| **商品编号** | **商品名称** | **单位** | **商品价格** |
| 1 | 挖掘机 | 台 | 1200000￥ |
| 2 | 冲击钻 | 个 | 230￥ |
| 3 | 铲车 | 辆 | 980000￥ |

这样设计，在很大程度上减小了数据库的冗余。如果要获取订单的商品信息，使用商品编号到商品信息表中查询即可。

因此可以总结判断的方法是：

第一步：找出数据表中所有的码。  
第二步：根据第一步所得到的码，找出所有的主属性。  
第三步：数据表中，除去所有的主属性，剩下的就都是非主属性了。  
第四步：查看是否存在非主属性对码的部分函数依赖。

第三范式

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。

则就是第三范式需要确保数据表中的每一列数据都和主键直接相关，而不能间接相关。

比如在设计一个订单数据表的时候，可以将客户编号作为一个外键和订单表建立相应的关系。而不可以在订单表中添加关于客户其它信息（比如姓名、所属公司等）的字段。如下面这两个表所示的设计就是一个满足第三范式的数据库表。

订单信息表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **订单编号** | **订单项目** | **负责人** | **业务员** | **订单数量** | **客户编号** |
| 001 | 挖掘机 | 刘明 | 李东明 | 1台 | 1 |
| 002 | 冲击钻 | 李刚 | 霍新峰 | 8个 | 2 |
| 003 | 铲车 | 郭新一 | 艾美丽 | 2辆 | 1 |

客户信息表

|  |  |  |  |
| --- | --- | --- | --- |
| **客户编号** | **客户名称** | **所属公司** | **联系方式** |
| 1 | 李聪 | 五一建设 | 13253661015 |
| 2 | 刘新明 | 个体经营 | 13285746958 |

这样在查询订单信息的时候，就可以使用客户编号来引用客户信息表中的记录，也不必在订单信息表中多次输入客户信息的内容，减小了数据冗余。

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。