十四、数据库设计范式

数据库设计的范式只是为数据库表的设计提供一个参考，告诉设计者如何把表设计的更加合理，具体表的设计还要根据实际情况而定。使用数据库范式可以让数据表更好的进行数据保存，但合理的设计在数据量很大的情况下也会影响性能，所以在开发中唯一称为设计宝典的是：在设计时尽量避免程序中出现多表关联查询；

# 第一范式：字段不可再分

所谓第一范式，就是指数据表中的数据字段不可再分；

例如：在superman用户下创建super\_member表

|  |
| --- |
| CREATE TABLE super\_member(  mid number(5) PRIMARY KEY,  name VARCHAR2(20),  contact VARCHAR2(100)  ); |

上表中联系方式设计不合理，联系方式包括多种，包括联系电话、email、联系地址等，而这些方式的格式不一定一致；故上表需要按照联系方式的不同，添加多个字段，从而符合第一范式；

|  |
| --- |
| CREATE TABLE super\_member(  mid number(5) PRIMARY KEY,  name VARCHAR2(20),  phone number(11),  email VARCHAR2(20),  address VARCHAR2(100)  ); |

**注：**

1. 一般情况下，姓名字段在国外需要分成名和姓，而在国内则不用；
2. 生日等日期一般保留整个年月日，而不用拆分；

所谓不可分割，指的是所有字段的所有数据类型可以用数据库提供的数据类型进行标示，而每个字段的数据类型都是固定的；

# 第二范式：多对多关系

第二范式：数据表中的非关键字段对候选关键字段存在部分函数依赖；

理解：

1）字段之间不存在函数关系；

例如：创建一个订单表，表中有价格，数量和总价字段，而总价=价格\*数量，存在函数关系，顾不符合第一范式；

|  |
| --- |
| CREATE TABLE business\_order(  order\_id NUMBER(5) PRIMARY KEY,  price NUMBER(5),  amount NUMBER(5),  total\_price NUMBER(10)  ); |

2）避免多个字段之间存在着多对多的关系；

例如，要实现一个学生选课系统，如果按照第一范式，则建表如下：

|  |
| --- |
| CREATE TABLE student\_course(  student\_id NUMBER(5) PRIMARY KEY,  student\_name VARCHAR(20) NOT NULL,  course\_name VARCHAR(20) NOT NULL,  credit NUMBER(5) NOT NULL,  score NUMBER(5)  ); |

上述表中存在以下几个问题：

* 数据重复：一门学生有多门课程，每增加一门课程，就要增加重复的学生信息；
* 数据更新过多：若要修改课程的某个信息，则需要将所有学生的该课程的信息都修改一遍，影响性能；
* 数据丢失：若有一门课程没有一个学生参加，则课程就从表中消失；

修改如下：将存在一对多关系的两个字段放在两张不同的表中；

|  |
| --- |
| CREATE TABLE student(  student\_id NUMBER(5) PRIMARY KEY,  student\_name VARCHAR(20) NOT NULL  );  CREATE TABLE course(  course\_id NUMBER(5) PRIMARY KEY,  course\_name VARCHAR(20) NOT NULL,  credit NUMBER(5) NOT NULL  );  CREATE TABLE score(  student\_id REFERENCES student(student\_id),  course\_id REFERENCES course(course\_id),  score NUMBER(5)  ); |

如此修改，学生和课程信息之间相互不影响，只有在进行分数查询时才会有关联；

# 第三范式：一对多关系

假如现在要创建一个数据库，实现一个学校有多个学生，用第一范式实现不了，用第二范式实现多对多关系，就需要用到第三范式；实现如下：

|  |
| --- |
| CREATE TABLE school(  school\_id NUMBER(5) PRIMARY KEY,  school\_name VARCHAR(20) NOT NULL  );  CREATE TABLE student(  student\_id NUMBER(5) PRIMARY KEY,  student\_name VARCHAR(20) NOT NULL,  school\_id REFERENCES school(school\_id)  ); |

第三范式是数据库设计中使用最多的范式；

**注意：数据库设计的范式只是为数据库表的设计提供一个参考，告诉设计者如何把表设计的更加合理，具体表的设计还要根据实际情况而定 ！**

### 第一范式（1NF）

所谓第一范式（1NF）是指在[关系模型](http://baike.baidu.com/view/176484.htm)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。

说明：在任何一个[关系数据库](http://baike.baidu.com/view/68348.htm)中，第一范式（1NF）是对[关系模式](http://baike.baidu.com/view/68347.htm)的设计基本要求，一般设计中都必须满足第一范式（1NF）。不过有些关系模型中突破了1NF的限制，这种称为非1NF的关系模型。换句话说，是否必须满足1NF的最低要求，主要依赖于所使用的[关系模型](http://baike.baidu.com/view/176484.htm)。

### 第二范式（2NF）

在1NF的基础上，非码属性必须完全依赖于码[在1NF基础上消除非主属性对主码的部分函数依赖]

第二范式（2NF）是在第一范式（1NF）的基础上建立起来的，即满足第二范式（2NF）必须先满足第一范式（1NF）。第二范式（2NF）要求数据库表中的每个实例或记录必须可以被唯一地区分。选取一个能区分每个实体的属性或属性组，作为实体的唯一标识。例如在员工表中的身份证号码即可实现每个一员工的区分，该身份证号码即为候选键，任何一个候选键都可以被选作主键。在找不到候选键时，可额外增加属性以实现区分，如果在员工关系中，没有对其身份证号进行存储，而姓名可能会在数据库运行的某个时间重复，无法区分出实体时，设计辟如ID等不重复的编号以实现区分，被添加的编号或ID选作主键。（该主键的添加是在ER设计时添加，不是建库时随意添加）

第二范式（2NF）**要求实体的属性完全依赖于主关键字。所谓完全依赖是指不能存在仅依赖主关键字一部分的属性**，如果存在，那么这个属性和主关键字的这一部分应该分离出来形成一个新的实体，新实体与原实体之间是一对多的关系。为实现区分通常需要为表加上一个列，以存储各个实例的唯一标识。简而言之，第二范式就是在第一范式的基础上属性完全依赖于主键。

### 第三范式（3NF）

在1NF基础上，**任何非主**[**属性**](http://baike.baidu.com/view/77730.htm)**不依赖于其它非主属性[在2NF基础上消除传递依赖]**

第三范式（3NF）是第二范式（2NF）的一个子集，即满足第三范式（3NF）必须满足第二范式（2NF）。简而言之，第三范式（3NF）要求一个关系中不包含已在其它关系已包含的非主关键字信息。例如，存在一个部门信息表，其中每个部门有部门编号（dept\_id）、部门名称、部门简介等信息。那么在员工信息表中列出部门编号后就不能再将部门名称、部门简介等与部门有关的信息再加入员工信息表中。如果不存在部门信息表，则根据第三范式（3NF）也应该构建它，否则就会有大量的数据冗余。简而言之，第三范式就是属性不依赖于其它非主属性，也就是在满足2NF的基础上，任何非主属性不得传递依赖于主属性。

### 巴斯-科德范式（BCNF）

Boyce-Codd Normal Form（巴斯-科德范式）

在1NF基础上，任何非主属性不能对主键子集依赖[在3NF基础上消除对主码子集的依赖]

巴斯-科德范式（BCNF）是第三范式（3NF）的一个子集，即满足巴斯-科德范式（BCNF）必须满足第三范式（3NF）。通常情况下，巴斯-科德范式被认为没有新的设计规范加入，只是对第二范式与第三范式中设计规范要求更强，因而被认为是修正第三范式，也就是说，它事实上是对第三范式的修正，使数据库冗余度更小。这也是BCNF不被称为第四范式的原因。某些书上，根据范式要求的递增性将其称之为第四范式是不规范，也是更让人不容易理解的地方。而真正的第四范式，则是在设计规范中添加了对多值及依赖的要求。

对于BCNF，在主码的任何一个[真子集](http://baike.baidu.com/view/1205.htm)都不能决定于主属性。关系中U主码，若U中的任何一个真子集X都不能决定于主属性Y，则该设计规范属性BCNF。例如：在关系R中，U为主码，A属性是主码中的一个属性，若存在A->Y,Y为主属性，则该关系不属性BCNF。

一般关系型数据库设计中，达到BCNF就可以了！

## 3范式应用实例[编辑](http://baike.baidu.com/link?url=DWO7F1OC5haMChTfAuGjxH7ETfhTcOmFfPhjapgnTCPSbnmdXSEjQb1Av1in1Tm4jS1x1VsLYbiWDyttNmoxDa)

下面以一个学校的学生系统为例分析说明，这几个范式的应用。

### 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](http://baike.baidu.com/view/14749.htm)、字符型、逻辑型、日期型等。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。

首先我们确定一下要设计的内容包括那些。学号、学生姓名、年龄、性别、课程名称、课程学分、系别、学科成绩，系办地址、系办电话等信息。为了简单我们暂时只考虑这些字段信息。我们对于这些信息，所关心的问题有如下几个方面。

学生有那些基本信息

学生选了那些课，成绩是什么?

每个课的学分是多少

学生属于那个系，系的基本信息是什么。

### 第二范式（2NF）

首先我们考虑，把所有这些信息放到一个表中（学号，学生姓名、年龄、性别、课程、课程学分、系别、学科成绩，系办地址、系办电话）下面存在如下的依赖关系。

(学号, 课程名称) → (姓名, 年龄, 成绩, 学分)

**问题分析**

因此不满足第二范式的要求，会产生如下问题

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。

更新异常：

1）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。

2）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。

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。

**解决方案**

把选课关系表SelectCourse改为如下三个表：

学生：Student（学号，姓名，年龄，性别，系别，系办地址、系办电话）；

课程：Course（课程名称,学分）；

选课关系：SelectCourse（学号，课程名称，成绩）。

### 第三范式（3NF）

接着看上面的学生表Student（学号，姓名，年龄，性别，系别，系办地址、系办电话），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，性别，系别，系办地址、系办电话

但是还存在下面的决定关系

（学号） → (系别）→（系办地点，系办电话）

即存在非关键字段"系办地点"、"系办电话"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。（数据的更新，删除异常这里就不分析了，可以参照2.1.1进行分析）

根据第三范式把学生关系表分为如下两个表就可以满足第三范式了：

学生：（学号，姓名，年龄，性别，系别）；

系别：（系别，系办地址、系办电话）。

上面的数据库表就是符合I，Ⅱ，Ⅲ范式的，消除了数据冗余、更新异常、插入异常和删除异常。