# 数据库的第一范式、第二范式、第三范式

 范式：英文名称是 Normal Form，它是英国人 E.F.Codd（关系数据库的老祖宗）在上个世纪70年代提出关系数据库模型后总结出来的，范式是关系数据库理论的基础，也是我们在设计数据库结构过程中所要遵循的规则和指导方法。目前有迹可寻的共有8种范式，依次是：1NF，2NF，3NF，BCNF，4NF，5NF，DKNF，6NF。通常所用到的只是前三个范式，即：第一范式（1NF），第二范式（2NF），第三范式（3NF）。下面就简单介绍下这三个范式。

## **第一范式（1NF）：**

强调的是列的原子性，即列不能够再分成其他几列。  
考虑这样一个表：【联系人】（姓名，性别，电话）  
如果在实际场景中，一个联系人有家庭电话和公司电话，那么这种表结构设计就没有达到 1NF。要符合 1NF 我们只需把列（电话）拆分，即：【联系人】（姓名，性别，家庭电话，公司电话）。1NF 很好辨别，但是 2NF 和 3NF 就容易搞混淆。

## 第二范式（2NF）：

首先是 1NF，另外包含两部分内容，一是表必须有一个主键；二是没有包含在主键中的列必须完全依赖于主键，而不能只依赖于主键的一部分。  
考虑一个订单明细表：【OrderDetail】（OrderID，ProductID，UnitPrice，Discount，Quantity，ProductName）。  
因为我们知道在一个订单中可以订购多种产品，所以单单一个 OrderID 是不足以成为主键的，主键应该是（OrderID，ProductID）。显而易见 Discount（折扣），Quantity（数量）完全依赖（取决）于主键（OderID，ProductID），而 UnitPrice，ProductName 只依赖于 ProductID。所以 OrderDetail 表不符合 2NF。不符合 2NF 的设计容易产生冗余数据。  
可以把【OrderDetail】表拆分为【OrderDetail】（OrderID，ProductID，Discount，Quantity）和【Product】（ProductID，UnitPrice，ProductName）来消除原订单表中UnitPrice，ProductName多次重复的情况。

## 第三范式（3NF）：

首先是 2NF，另外非主键列必须直接依赖于主键，不能存在传递依赖。即不能存在：非主键列 A 依赖于非主键列 B，非主键列 B 依赖于主键的情况。  
考虑一个订单表【Order】（OrderID，OrderDate，CustomerID，CustomerName，CustomerAddr，CustomerCity）主键是（OrderID）。  
其中 OrderDate，CustomerID，CustomerName，CustomerAddr，CustomerCity 等非主键列都完全依赖于主键（OrderID），所以符合 2NF。不过问题是 CustomerName，CustomerAddr，CustomerCity 直接依赖的是 CustomerID（非主键列），而不是直接依赖于主键，它是通过传递才依赖于主键，所以不符合 3NF。  
通过拆分【Order】为【Order】（OrderID，OrderDate，CustomerID）和【Customer】（CustomerID，CustomerName，CustomerAddr，CustomerCity）从而达到 3NF。  
第二范式（2NF）和第三范式（3NF）的概念很容易混淆，区分它们的关键点在于，2NF：非主键列是否完全依赖于主键，还是依赖于主键的一部分；3NF：非主键列是直接依赖于主键，还是直接依赖于非主键列。

# [mysql 数据库的设计三范式](https://www.cnblogs.com/xiaozengzeng/p/10720226.html)

## 三范式

1NF:字段不可分;

2NF:有主键，非主键字段依赖主键;

3NF:非主键字段不能相互依赖;

解释:

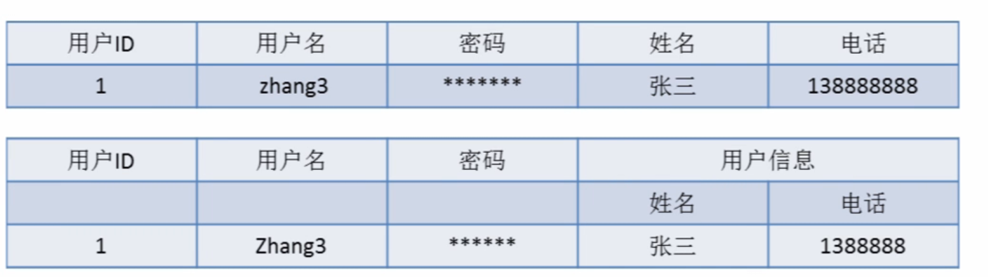
1NF:原子性 字段不可再分,否则就不是关系数据库;

2NF:唯一性 一个表只说明一个事物;

3NF:每列都与主键有直接关系，不存在传递依赖;

#### 第一范式（1NF）

即表的列的具有原子性,不可再分解，即列的信息，不能分解, 只要数据库是关系型数据库(mysql/oracle/db2/informix/sysbase/sql server)，就自动的满足1NF。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。如果实体中的某个属性有多个值时，必须拆分为不同的属性 。通俗理解即一个字段只存储一项信息。



关系型数据库: mysql/oracle/db2/informix/sysbase/sql server 非关系型数据库: (特点: 面向对象或者集合) NoSql数据库: MongoDB/redis(特点是面向文档)

#### 第二范式（2NF）

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行必须可以被惟一地区分。为实现区分通常需要我们设计一个主键来实现(这里的主键不包含业务逻辑)。

即满足第一范式前提，当存在多个主键的时候，才会发生不符合第二范式的情况。比如有两个主键，不能存在这样的属性，它只依赖于其中一个主键，这就是不符合第二范式。通俗理解是任意一个字段都只依赖表中的同一个字段。（涉及到表的拆分）

看下面的学生选课表：

| **学号** | **课程** | **成绩** | **课程学分** |
| --- | --- | --- | --- |
| 10001 | 数学 | 100 | 6 |
| 10001 | 语文 | 90 | 2 |
| 10001 | 英语 | 85 | 3 |
| 10002 | 数学 | 90 | 6 |
| 10003 | 数学 | 99 | 6 |
| 10004 | 语文 | 89 | 2 |

表中主键为 （学号，课程），我们可以表示为 (学号，课程) -> (成绩，课程学分)， 表示所有非主键列 (成绩，课程学分)都依赖于主键 (学号，课程)。 但是，表中还存在另外一个依赖：（课程）->(课程学分）。这样非主键列 ‘课程学分‘ 依赖于部分主键列 ’课程‘， 所以上表是不满足第二范式的。

我们把它拆成如下2张表：

学生选课表：

| **学号** | **课程** | **成绩** |
| --- | --- | --- |
| 10001 | 数学 | 100 |
| 10001 | 语文 | 90 |
| 10001 | 英语 | 85 |
| 10002 | 数学 | 90 |
| 10003 | 数学 | 99 |
| 10004 | 语文 | 89 |

课程信息表：

| **课程** | **课程学分** |
| --- | --- |
| 数学 | 6 |
| 语文 | 3 |
| 英语 | 2 |

那么上面2个表，学生选课表主键为（学号，课程），课程信息表主键为（课程），表中所有非主键列都完全依赖主键。不仅符合第二范式，还符合第三范式。

再看这样一个学生信息表：

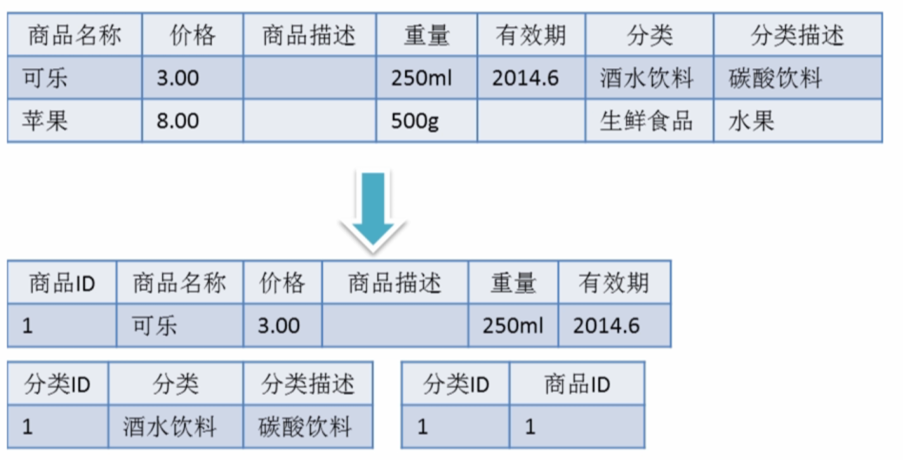
| **学号** | **姓名** | **性别** | **班级** | **班主任** |
| --- | --- | --- | --- | --- |
| 10001 | 张三 | 男 | 一班 | 小王 |
| 10002 | 李四 | 男 | 一班 | 小王 |
| 10003 | 王五 | 男 | 二班 | 小李 |
| 10004 | 张小三 | 男 | 二班 | 小李 |

上表中，主键为：（学号），所有字段 （姓名，性别，班级，班主任）都依赖与主键（学号），不存在对主键的部分依赖。所以是满足第二范式。

#### 第三范式（3NF）

满足第三范式（3NF）必须先满足第二范式（2NF）。简而言之，第三范式（3NF）要求一个数据库表中不包含已在其它表中已包含的非主键字段。就是说，表的信息，如果能够被推导出来，就不应该单独的设计一个字段来存放(能尽量外键join就用外键join)。很多时候，我们为了满足第三范式往往会把一张表分成多张表。

即满足第二范式前提，如果某一属性依赖于其他非主键属性，而其他非主键属性又依赖于主键，那么这个属性就是间接依赖于主键，这被称作传递依赖于主属性。 通俗解释就是一张表最多只存两层同类型信息。



#### 反三范式

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，提高读性能，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，减少了查询时的关联，提高查询效率，因为在数据库的操作中查询的比例要远远大于DML的比例。但是反范式化一定要适度，并且在原本已满足三范式的基础上再做调整的。