# 规范化－数据库设计原则

## 摘要

关系型数据库是当前广泛应用的数据库类型，关系数据库设计是对数据进行组织化和结构化的过程，核心问题是关系模型的设计。对于数据库规模较小的情况，我们 可以比较轻松的处理数据库中的表结构。然而，随着项目规模的不断增长，相应的数据库也变得更加复杂，关系模型表结构更为庞杂，这时我们往往会发现我们写出 来的SQL语句的是很笨拙并且效率低下的。更糟糕的是，由于表结构定义的不合理，会导致在更新数据时造成数据的不完整。因此，就有必要学习和掌握数据库的 规范化流程，以指导我们更好的设计数据库的表结构，减少冗余的数据，借此可以提高数据库的存储效率，数据完整性和可扩展性。

简洁、结构明晰的表结构对数据库的设计是相当重要的。规范化的表结构设计，在以后的数据维护中，不会发生插入（insert）、删除 （delete）和更新（update）时的异常。反之，数据库表结构设计不合理，不仅会给数据库的使用和维护带来各种各样的问题，而且可能存储了大量不 需要的冗余信息，浪费系统资源。

要设计规范化的数据库，就要求我们根据数据库设计范式――也就是数据库设计的规范原则来做。但是一些相关材料上提到的范式设计，往往是给出一大堆的公式，这给设计者的理解和运用造成了一定的困难。

## 规范化

在设计和操作维护数据库时，关键的步骤就是要确保数据正确地分布到数据库的表中。 使用正确的数据结构，不仅便于对数据库进行相应的存取操作，而且可以极大地简化应用程序的其他内容（查询、窗体、报表、代码等）。正确进行表设计的正式名称就是"数据库规范化"。

## 数据冗余

数据应该尽可能少地冗余，这意味着重复数据应该减少到最少。比如说，一个部门雇员的电话不应该被存储在不同的表中， 因为这里的电话号码是雇员的一个属性。如果存在过多的冗余数据，这就意味着要占用了更多的物理空间，同时也对数据的维护和一致性检查带来了问题，当这个员 工的电话号码变化时，冗余数据会导致对多个表的更新动作，如果有一个表不幸被忽略了，那么就可能导致数据的不一致性。

## 规范化实例

CREATE TABLE `SAMPLE` (

`PRJNUM` int(11) NOT NULL ,

`PRJNAME` varchar(200) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL ,

`EMYNUM` int(11) NOT NULL ,

`EMYNAME` varchar(200) CHARACTER SET utf8 COLLATE utf8\_general\_ci NOT NULL ,

`SALCATEGORY` char(1) CHARACTER SET utf8 COLLATE utf8\_general\_ci NOT NULL ,

`SALPACKAGE` int(11) NOT NULL ,

PRIMARY KEY (`PRJNUM`, `EMYNUM`)

)

ENGINE=InnoDB

DEFAULT CHARACTER SET=utf8 COLLATE=utf8\_general\_ci

ROW\_FORMAT=COMPACT

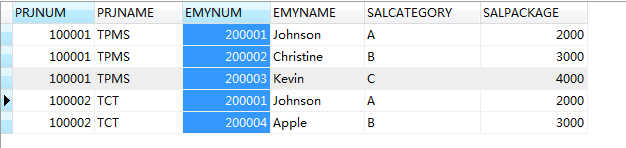
Insert into SAMPLE(PRJNUM, PRJNAME, EMYNUM, EMYNAME, SALCATEGORY, SALPACKAGE)

values(100001, 'TPMS', 200001, 'Johnson', 'A', 2000), (100001, 'TPMS', 200002,

'Christine', 'B', 3000), (100001, 'TPMS', 200003, 'Kevin', 'C', 4000), (100002,

'TCT', 200001, 'Johnson', 'A', 2000), (100002, 'TCT', 200004, 'Apple', 'B',

3000);

第一范式 1NF

参照范式的定义，考察上表，我们发现，这张表已经满足了第一范式的要求。

1、因为这张表中字段都是单一属性的，不可再分；

2、而且每一行的记录都是没有重复的；

3、存在主属性，而且所有的属性都是依赖于主属性；

4、所有的主属性都已经定义

事实上在当前所有的关系数据库管理系统（DBMS）中，都已经在建表的时候强制满足第一范式。因此，这张SAMPLE表已 经是一张满足第一范式要求的表。考察表1-1，我们首先要找出主键。可以看到，属性对<Project Number, Employee Number>是主键，其他所有的属性都依赖于该主键。

## 从一范式转化到二范式

根据第二范式的定义，转化为二范式就是消除部分依赖。

考察表1-1，我们可以发现，非主属性<Project Name>部分依赖于主键中的<Project Number>; 非主属性<Employee Name>，<Salary Category>和<Salary package>都部分依赖于主键中的<Employee Number>；

表1-1的形式，存在着以下潜在问题：

1． 数据冗余：每一个字段都有值重复；

2． 更新异常：比如<Project Name>字段的值，比如对值"TPMS"了修改，那么就要一次更新该字段的多个值；

3． 插入异常：如果新建了一个Project，名字为TPT, 但是还没有Employee加入，那么<Employee Number>将会空缺，而该字段是主键的一部分，因此将无法插入记录；

4． 删除异常：如果一个员工 200003, Kevin 离职了，要将该员工的记录从表中删除，而此时相关的Salary信息 C 也将丢失, 因为再没有别的行纪录下 Salary C的信息。

因此，我们需要将存在部分依赖关系的主属性和非主属性从满足第一范式的表中分离出来，形成一张新的表，而新表和旧表之间是一对多的关系。由此，我们得到：

CREATE TABLE `Project` (

`PRJNUM` int(11) NOT NULL ,

`PRJNAME` varchar(200) CHARACTER SET utf8 COLLATE utf8\_general\_ci NULL DEFAULT NULL ,

PRIMARY KEY (`PRJNUM`)

)

ENGINE=InnoDB

DEFAULT CHARACTER SET=utf8 COLLATE=utf8\_general\_ci

ROW\_FORMAT=COMPACT

CREATE TABLE `EMPLOYEE` (

`EMYNUM` int(11) NOT NULL ,

`EMYNAME` varchar(200) CHARACTER SET utf8 COLLATE utf8\_general\_ci NOT NULL ,

`SALCATEGORY` char(1) CHARACTER SET utf8 COLLATE utf8\_general\_ci NOT NULL ,

`SALPACKAGE` int(11) NULL DEFAULT NULL ,

PRIMARY KEY (`EMYNUM`)

)

ENGINE=InnoDB

DEFAULT CHARACTER SET=utf8 COLLATE=utf8\_general\_ci

ROW\_FORMAT=COMPACT

;

CREATE TABLE `PRJ\_EMY` (

`PRJNUM` int(11) NOT NULL ,

`EMYNUM` int(11) NOT NULL ,

PRIMARY KEY (`PRJNUM`, `EMYNUM`)

)

ENGINE=InnoDB

DEFAULT CHARACTER SET=utf8 COLLATE=utf8\_general\_ci

ROW\_FORMAT=COMPACT

;

## 从二范式转化到三范式

考察表前面生成的三张表，我们发现，表1-3存在传递依赖关系，即：关键字段< Employee Number > --> 非关键字段< Salary Category > -->非关键字段< Salary Package >。而这是不满足三范式的规则的，存在以下的不足：

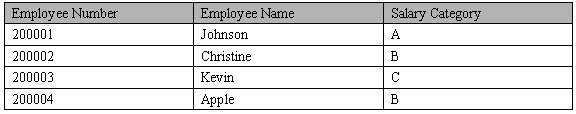
1、 数据冗余：<Salary Category>和<Salary Package>的值有重复；

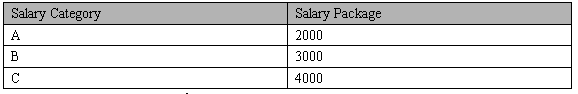
2、 更新异常：有重复的冗余信息，修改时需要同时修改多条记录，否则会出现数据不一致的情况；

3、 删除异常：同样的，如果员工 200003 Kevin 离开了公司，会直接导致 Salary C 的信息的丢失。

Delete from EMPLOYEE where EMYNUM = 200003  
Select distinct SALCATEGORY, SALPACKAGE from EMPLOYEE

因此，我们需要继续进行规范化的过程，把表1-3拆开，我们得到：





这时候如果 200003 Kevin 离开公司，我们只需要从表 1-5 中删除他就可以了， 存在于表1-6中的Salary C信息并不会丢失。但是我们要注意到除了表 1-5 中存在 Kevin 的信息之外， 表1-4中也存在 Kevin 的信息， 这很容易理解， 因为 Kevin 参与了项目 100001， TPMS， 所以当然也要从中删除。

至此，我们将表1-1经过规范化步骤，得到四张表，满足了三范式的约束要求，数据冗余、更新异常、插入异常和删除异常。

在三范式之上，还存在着更为严格约束的BC范式和四范式，但是这两种形式在商业应用中很少用到，在绝大多数情况下，三范式已经满足了数据库表规范化的要求，有效地解决了数据冗余和维护操作的异常问题。