范式

范式：Normal Farmat，是为了解决数据的存储和优化问题。

在数据存储之后，凡是能够通过关系寻找出来的数据，坚决不再重复存储，范式的终极目标是减少数据冗余。

范式是一种分层结构的规范，共 6 层，分别为1NF、2NF、3NF、4NF、5NF和6NF，每一层都比上一层严格，若要满足下一层范式，其前提是先满足上一层范式。其中，1NF是最底层的范式，6NF为最高层的范式，也最严格。

MySQL 数据库属于关系型数据库，其存储数据的时候有些浪费空间，但也致力于节省空间，这就与范式想要解决的问题不谋而合，因此在设计数据库的时候，大都会利用范式来指导设计。但是数据库不单是要解决存储空间的问题，还要保证效率的问题，而范式只为解决存储空间的问题，所以数据库的设计又不能完全按照范式的要求来实现，因此在一般情况下，只需要满足前三种范式即可。

此外，咱们需要知道：范式在数据库的设计中是有指导意义的，但不是强制规范。

1NF

第一范式：在设计表存储数据的时候，如果表中设计的字段存储的数据，在取出来使用之前还需要额外的处理（拆分），那么表的设计就不满足第一范式，第一范式要求字段的数据具有原子性，不可再分。

例如，咱们设计一个「学校假期时间表」，如下所示：

表 1：学校假期时间表

| ID(P) | 学校名称 | 起始日期，结束日期 |
| --- | --- | --- |
| 1 | 哈尔滨工业大学 | 20170625，20170903 |
| 2 | 浙江大学 | 20170630，20170901 |

观察上表，咱们会发现表1的设计并没有什么问题，但是如果需求是查询各学校开始放假的日期呢？那显然上表的设计并不满足1NF，数据不具有原子性。对于此类问题，解决的方案就是将表1进行拆分：

表 2：拆分后的表 1

| ID(P) | 学校名称 | 起始日期 | 结束日期 |
| --- | --- | --- | --- |
| 1 | 哈尔滨工业大学 | 20170625 | 20170903 |
| 2 | 浙江大学 | 20170630 | 20170901 |

2NF

第二范式：在数据表的设计过程中，如果有复合主键（多字段主键），且表中有字段并不是由整个主键来确定，而是依赖复合主键中的某个字段（主键的部分），也就是说存在字段依赖主键的部分的问题（称之为部分依赖），第二范式就是要解决表设计中不允许出现部分依赖。

例如，咱们设计一个「教室授课表」，如下所示：

表 3：教室授课表

| 教师(P) | 性别 | 课程 | 授课地点(P) |
| --- | --- | --- | --- |
| 许仙 | 男 | 《如何追到心爱的女孩》 | 杭州西湖 |
| 白娘子 | 女 | 《论女人的恋爱修养》 | 雷峰塔 |
| 白娘子 | 女 | 《如何打赢与和尚之间的持久战》 | 金山寺 |

观察上表，咱们会发现：教师不能作为独立的主键，需要与授课地点相结合才能作为主键（复合主键，每个教师的某个课程只能在固定的地点上），其中性别依赖于具体的教师，而课程依赖于授课地点，这就出现了表的字段依赖于部分主键的问题，从而导致不满足第二范式。

解决方案 1：将教师和性别，课程和授课地点，分成两张单独的表；

解决方案 2：取消复合主键，使用逻辑主键。

在此，咱们采用 方案 2 的解决方法，即取消复合主键，使用逻辑主键。

| ID(P) | 教师 | 性别 | 课程 | 授课地点 |
| --- | --- | --- | --- | --- |
| 1 | 许仙 | 男 | 《如何追到心爱的女孩》 | 杭州西湖 |
| 2 | 白娘子 | 女 | 《论女人的恋爱修养》 | 雷峰塔 |
| 3 | 白娘子 | 女 | 《如何打赢与和尚之间的持久战》 | 金山寺 |

3NF

第三范式：需要满足第一范式和第二范式，理论上讲，每张表中的所有字段都应该直接依赖主键（逻辑主键，代表是业务主键），如果表设计中存在一个字段，并不直接依赖主键，而是通过某个非主键字段依赖，最终实现主键依赖（把这种不是直接依赖主键，而是依赖非主键字段的依赖关系，称之为传递依赖），第三范式就是要解决表设计中出现传递依赖的问题。

以上述的添加逻辑主键后的 表3 为例：

| ID(P) | 教师 | 性别 | 课程 | 授课地点 |
| --- | --- | --- | --- | --- |
| 1 | 许仙 | 男 | 《如何追到心爱的女孩》 | 杭州西湖 |
| 2 | 白娘子 | 女 | 《论女人的恋爱修养》 | 雷峰塔 |
| 3 | 白娘子 | 女 | 《如何打赢与和尚之间的持久战》 | 金山寺 |

在以上表的设计中，性别依赖教师，教师依赖主键；课程依赖授课地点，授课地点依赖主键，因此性别和课程都存在传递依赖的问题。

解决方案：将存在传递依赖的字段，以及依赖的字段本身单独取出来，形成一个单独的表，然后在需要使用对应的信息的时候，把对应的实体表的主键添加进来。

表 4：教师表

| TEACHER\_ID(P) | 教师 | 性别 |
| --- | --- | --- |
| 1 | 许仙 | 男 |
| 2 | 白娘子 | 女 |
| 3 | 白娘子 | 女 |

表 5：授课地点表

| ADDRESS\_ID(P) | 课程 | 授课地点 |
| --- | --- | --- |
| 1 | 《如何追到心爱的女孩》 | 杭州西湖 |
| 2 | 《论女人的恋爱修养》 | 雷峰塔 |
| 3 | 《如何打赢与和尚之间的持久战》 | 金山寺 |

表 6：进行处理后的表

| ID(P) | TEACHER\_ID | ADDRESS\_ID |
| --- | --- | --- |
| 1 | 1 | 1 |
| 2 | 2 | 2 |
| 3 | 3 | 3 |

在观察上述 表 4 和 表 5，咱们会发现TEACHER\_ID等价于教师且ADDRESS\_ID等价于授课地点，因此其逻辑主键并没有什么实际的限制意义，咱们只需要看其具体代表的业务主键即可。咱们之所以使用逻辑主键，是因为：逻辑主键可以实现自动增长，并且数字传递比较方便，而且有利于节省空间。

逆规范化

在某些特定的环境中（例如淘宝数据库），在设计表的时候，如果一张表中有几个字段是需要从另外的表中去获取数据，理论上讲，的确可以获得想要的数据，但是相对来说，其效率低会一点。此时为了提高查询效率，咱们会刻意的在某些表中，不去保存另外一张表的主键（逻辑主键），而是直接保存想要存储的数据信息，这样的话，在查询数据的时候，这张表就可以直接提供咱们想要的数据，而不需要多表查询，但是这样做会导致数据冗余。

实际上，逆规范化是磁盘利用率和效率之间的对抗。