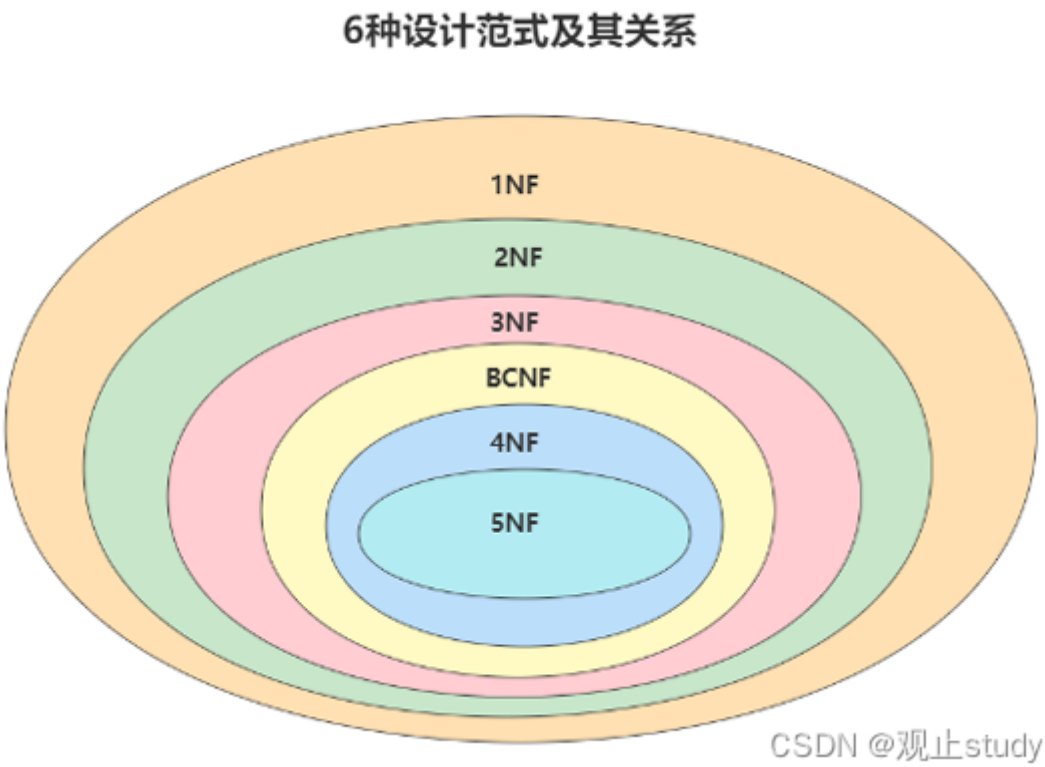
【MySQL】范式 (十五)

> 🚚MySQL学习·第十五站~
> 🚩本文已收录至专栏：MySQL通关路
> ❤️‍🩹文末附全文思维导图，感谢各位点赞收藏支持~
> ⭐学习汇总贴，超详细思维导图：【MySQL】学习汇总(完整思维导图)

一.引入

在关系型数据库中，关于数据表设计的基本原则、规则就称为范式，英文名称为`Normal Form`，简称`NF`。可以理解为，**一张数据表的设计结构需要满足的某种设计标准的级别**。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。

目前关系型数据库有六种常见范式，按照范式级别，从**低到高**分别是：**第一范式**（`1NF`）、**第二范式**（`2NF`）、**第三范式**（`3NF`）、**巴斯-科德范式**（`BCNF`）、**第四范式**(`4NF`）和**第五范式**（`5NF`，又称完美范式）。



数据库的范式**设计越高阶，冗余度就越低**，同时**高阶的范式一定符合低阶范式的要求**，例如满足最低要求的范式是第一范式（1NF）,在第一范式的基础上进一步满足更多的规范要求则称之为第二范式（2NF）。

一般来说，在关系型数据库设计中，最高也就遵循到`BCNF`，普遍还是`3NF`，这是因为越高阶复杂度也会越高并且会影响一定的性能。有时为了提高某些查询性能，我们还需要违背范式规则，也就是**反规范化**。

二. 键相关概念

范式的定义会涉及到主键和候选键，数据库中的**键(Key)由一个或者多个属性**组成。常用的几种键和属性的定义如下:

- `超键`：能**唯一标识元组的属性集**叫做超键。
- `候选键`：如果超键**不包含多余的属性**，那么这个超键就是候选键。
- `主键`：用户可以从**候选键中选择一个**作为主键。
- `外键`：如果数据表R1中的某属性集不是R1的主键，而**是另一个数据表R2的主键**，那么这个属性集就是数据表R1的外键。
- `主属性`：**包含在任一候选键中的属性**称为主属性。
- `非主属性`：与主属性相对，指的是**不包含在任何一个候选键中的属性**。

通常，我们也将**候选键称之为“码”**，把**主键也称为“主码”**。因为键可能是由多个属性组成的，针对单个属性，我们还可以用主属性和非主属性来进行区分。

示例： 有如下两个表：

球员表(player)：球员编号 | 姓名 | 身份证号 | 年龄 | 球队编号

球队表(team)：球队编号 | 主教练 | 球队所在地

- `超键`：对于球员表来说，超键就是包括球员编号或者身份证号的任意组合，比如（球员编号） （球员编号，姓名）（身份证号，年龄）等，因为这两个字段与任意属性组合都能唯一标识一名球员
- `候选键`：最小的超键组合，对于球员表来说，候选键就是（球员编号）或者（身份证号）
- `主键`：我们自己选定，也就是从候选键中任意选择一个，比如（球员编号）
- `外键`：球员表中的球队编号不是球员表的主键，而是球队表的主键。
- `主属性`：在球员表中，包含在任一候选键中的属性，例如（球员编号）（身份证号）
- `非主属性`：在球员表中，不包含在任一候选键中的属性，例如（姓名） （年龄） （球队编号）

三.范式

(1) 第一范式

第一范式主要是确保数据表中**每个字段的值必须具有原子性**，也就是说数据表中**每个字段的值为不可再次拆分的最小数据单元**。例如，我们在设计某个字段的时候，对于字段x来说，不能把字段x拆分成字段X-1和字段X-2。

示例：

下述user 表的设计不符合第一范式，其中，`user_info`字段为用户信息，可以进一步拆分成更小粒度的字段，不符合数据库设计对第一范式的要求。

字段名称	字段类型	是否是主键	说明
id	INT	是	主键id
username	VARCHAR(30)	否	用户名
password	VARCHAR(50)	否	密码
user_info	VARCHAR(255)	否	用户信息 (包含真实姓名、电话、住址)

需要将user_info进行如下拆分：

字段名称	字段类型	是否是主键	说明
id	INT	是	主键id
username	VARCHAR(30)	否	用户名
password	VARCHAR(50)	否	密码
real_name	VARCHAR(30)	否	真实姓名
phone	VARCHAR(12)	否	联系电话
address	VARCHAR(100)	否	家庭住址

注意事项：

属性的原子性是主观的。例如，`user`表中的`address`地址字段，我们是否需要再拆分为省、市、县三个字段？答案取决于应用程序。如果应用程序需要分别处理不同级别的地址，则有必要把它们分开。否则，不需要。

姓名	年龄	地址
张三	20	广东省广州市三元里78号
李四	24	广东省深圳市龙华新区

根据实际场景进行拆分

姓名	年龄	省	市	地址
张三	20	广东	广州	三元里78号
李四	24	广东	深圳	龙华新区

(2) 第二范式

第二范式要求，在满足第一范式的基础上，还要满足**数据表里的每一条数据记录，都是可唯一标识的**，即必须有超键、候选键、主键。而且**所有非主键字段，都必须完全依赖主键，不能只依赖主键的一部分**。如果知道主键的所有属性的值，就可以检索到任何元组(行)的任何属性的任何值。

示例一：

在成绩表（学号，课程号，成绩）关系中，学号不能决定成绩，课程号也不能决定成绩，只有（学号，课程号）一起才可以决定成绩，所以`（学号，课程号）→ 成绩`就是**完全依赖关系**。

举例二：

在比赛表（球员编号、姓名、年龄、比赛编号、比赛时间、比赛场地、得分）关系中，其主键为（球员编号，比赛编号），我们可以通过主键来决定如下的关系：`（球员编号，比赛编号）→（姓名，年龄，比赛时间，比赛场地，得分）`，但是这个数据表不满足第二范式，因为数据表中的字段之间还存在着如下的对应关系：`（球员编号）→（姓名，年龄）`和`（比赛编号）→（比赛时间，比赛场地）`。非主键字段并没有完全依赖于主键，而是只依赖于主键的一部分。这将会导致如下问题：

1. **数据冗余**：如果一个球员可以参加 m 场比赛，那么球员的姓名和年龄就重复了 m-1 次。一个比赛也可能会有 n 个球员参加，比赛的时间和地点就重复了 n-1 次。
2. **插入异常**：如果我们想要添加一场新的比赛，但是这时还没有确定参加的球员都有谁，那么就没法插入。
3. **删除异常**：如果我要删除某个球员编号，如果没有单独保存比赛表的话，就会同时把比赛信息删除掉。
4. **更新异常**：如果我们调整了某个比赛的时间，那么数据表中所有这个比赛的时间都需要进行调整，否则就会出现同一场比赛时间不同的情况。

为了避免出现上述的情况，我们可以把球员比赛表设计为下面的三张表。

表名	属性（字段）
球员 player 表	球员编号、姓名和年龄等属性
比赛 game 表	比赛编号、比赛时间和比赛场地等属性
球员比赛关系 player_game 表	球员编号、比赛编号和得分等属性

这样的话，每张数据表都符合第二范式，也就避免了上述异常情况的发生。

(3) 第三范式

第三范式是在第二范式的基础上，确保数据表中的**每一个非主键字段都和主键字段直接相关**，也就是说，要求数据表中的**所有非主键字段不能依赖于其他非主键字段**。(即，不能存在非主属性A依赖于非主属性B，非主属性B依赖于主键c的情况，即存在"A→B→c""的决定关系）通俗地讲，该规则的意思是**所有非主键属性之间不能有依赖关系，必须相互独立**。

举例:

有如下商品信息表:

字段名称	字段类型	是否是主键	说明
id	INT	是	商品主键id （主键）
category_id	INT	否	非主属性 商品类别id
category_name	VARCHAR(30)	否	非主属性 商品类别名称
goods_name	VARCHAR(30)	否	商品名称
price	DECIMAL(10,2)	否	商品价格

其中主键为商品id，而表中商品类别名称依赖于商品类别编号，不符合第三范式要求。我们需要进行如下拆分：

商品类别表:

字段名称	字段类型	是否是主键	说明
id	INT	是	商品类别主键id
category_name	VARCHAR(30)	否	商品类别名称

商品表:

字段名称	字段类型	是否是主键	说明
id	INT	是	商品主键id
category_id	VARCHAR(30)	否	商品类别id
goods_name	VARCHAR(30)	否	商品名称
price	DECIMAL(10,2)	否	商品价格

商品表通过商品类别id字段（category_id）与商品类别表goods_category进行关联，如此所有非主键属性便是相互独立，满足第三范式要求。

(4) 小结

由于我们在大多数情况下只需要遵循到第三范式，在此做一番小结，后续几个范式了解即可。

范式	要求
第一范式(1NF)	确保每列保持`原子性`，为不可再分的最小数据单元。
第二范式(2NF)	确保每列都和主键`完全依赖`，尤其在复合主键的情况下，非主键部分不应该依赖于部分主键。
第三范式(3NF)	确保每列都和主键列`直接相关`，而不是间接相关

优点:

数据的标准化有助于消除数据库中的数据冗余以及几种异常，第三范式(3NF）通常被认为在性能、扩展性和数据完整性方面达到了最好的平衡。

缺点：

范式的使用，可能降低查询的效率。因为范式等级越高，设计出来的数据表就越多、越精细，数据的冗余度就越低，进行数据查询的时候就可能需要关联多张表，这不但代价昂贵，也可能使一些索引策略无效。

特别说明：

范式只是提出了设计的标准，实际上设计数据表时，未必一定要符合这些标准。开发中，我们会出现为了性能和读取效率违反范式化的原则，通过增加少量的冗余或重复的数据来提高数据库的读性能，减少关联查询,join表的次数，实现空间换取时间的目的。范式本身没有优劣之分，只有适用场景不同。没有完美的设计，只有合适的设计。

(5) 巴斯-科德范式

巴斯-科德范式(Boyce-Codd NormalForm)被认为没有新的设计规范加入，只是对第三范式中设计规范要求更强，使得数据库冗余度更小。所以，也称为是修正的第三范式，或扩充的第三范式，而不被称为第四范式。

若一个关系达到了第三范式，并且它**只有一个候选键**，或者它的**每个候选键都是单属性**，则该关系自然达到BC范式。

示例：有如下仓库仓库管理关系表

仓库名	管理员	物品名	数量
北京仓	张三	iphone XR	10
北京仓	张三	iphone 7	20
上海仓	李四	iphone 7p	30
上海仓	李四	iphone 8	40

在这个表中，一个仓库只有一个管理员，同时一个管理员也只管理一个仓库。 仓库名决定了管理员，管理员也决定了仓库名，同时（仓库名，物品名）的属性集合可以决定数量这个属性。由此我们可以得出：

候选键：（管理员，物品名）和（仓库名，物品名）

主键：候选键中的任意一个，如（仓库名，物品名）。

主属性：包含在任一候选键中的属性，也就是仓库名，管理员和物品名。

非主属性：数量这个属性

我们先来判断一下是否满足第三范式， 首先，数据表每个属性都是原子性的，符合 1NF 的要求； 其次，数据表中非主属性”数量“都与候选键全部依赖，（仓库名，物品名）决定数量，（管理员，物品名）决定数量，符合 2NF 的要求； 最后，数据表中的非主属性，不传递依赖于候选键。因此符合 3NF 的要求。

存在的问题：

虽然数据表已经符合了 3NF 的要求，但是还是存在一定的问题：

1. **插入异常**：增加一个仓库，但是还没有存放任何物品。根据数据表实体完整性的要求，主键不能有空值，因此会出现插入异常；
2. **更新异常**：如果某个仓库更换了管理员，我们就可能需要同步修改数据表中的多条记录；

3. **删除异常**：如果仓库里的商品都卖空了，那么此时仓库名称和相应的管理员名称也会随之被删除。

问题解决：

首先我们需要确认造成异常的原因：主属性仓库名对于候选键（管理员，物品名）是部分依赖的关系， 这样就有可能导致上面的异常情况。因此引入BCNF，它在 3NF 的基础上消除了主属性对候选键的部分依赖或者传递依赖关系。（如果在关系R中，U为主键，A属性是主键的一个属性，若存在A->Y，Y为主属性，则该关系不属于 BCNF）

根据 BCNF 的要求，我们需要把仓库管理关系表拆分成下面这样：

仓库表：（仓库名，管理员）

库存表：（仓库名，物品名，数量）

这样就不存在主属性对于候选键的部分依赖或传递依赖，上面数据表的设计就符合 BCNF。

(6) 第四范式

相关概念:

- `多值依赖`：即属性之间的一对多关系，记为K→→A。
- `函数依赖`：事实上是单值依赖，所以不能表达属性值之间的一对多关系。
- `平凡的多值依赖`：全集U=K+A，一个K可以对应于多个A，即K→→A。此时整个表就是一组一对多关系。
- `非平凡的多值依赖`：全集U=K+A+B，一个K可以对应于多个A，也可以对应于多个B，A与B互相独立，即K→→A，K→→B。整个表有多组一对多关系，且有:“一"部分是相同的属性集合，“多"部分是互相独立的属性集合。

第四范式即在满足巴斯-科德范式(BCNF)的基础上，消除非平凡且非函数依赖的多值依赖（即把同一表内的多对多关系删除)。

示例：

在职工表(职工编号，职工孩子姓名，职工选修课程)中。同一个职工可能会有多个职工孩子姓名。同样，同一个职工也可能会有多个职工选修课程，即这里存在着多值事实，不符合第四范式。 如果要符合第四范式，只需要将上表分为两个表，使它们只有一个多值事实，例如：职工表一(职工编号，职工孩子姓名)，职工表二(职工编号，职工选修课程)，两个表都只有一个多值事实，所以符合第四范式。

(7) 第五范式

在满足第四范式(4NF)的基础上，消除不是由候选键所蕴含的连接依赖。如果关系模式R中的每一个连接依赖均由R的候选键所隐含，则称此关系模式符合第五范式。函数依赖是多值依赖的一种特殊的情况，而多值依赖实际上是连接依赖的一种特殊情况。但连接依赖不像函数依赖和多值依赖可以由语义直接导出，而是在关系连接运算时才反映出来。存在连接依赖的关系模式仍可能遇到数据冗余及插入、修改、删除异常等问题。
第五范式处理的是无损连接问题，这个范式**基本没有实际意义，了解即可**，因为无损连接很少出现，而且难以察觉。

四.反范式

(1) 概述

有的时候不能简单按照规范要求设计数据表，因为有的数据看似冗余，其实对业务来说十分重要。这个时候，我们就要遵循业务优先的原则，首先满足业务需求，再尽量减少冗余。
例如：如果数据库中的数据量比较大，系统的UV和PV访问频次比较高，若完全按照MySQL的三大范式设计数据表，读数据时会产生大量的关联查询，在一定程度上会影响数据库的读性能。如果我们想对查询效率进行优化，反范式优化也是一种优化思路。此时，可以通过在数据表中**增加冗余字段来提高数据库的读性能**。

(2) 示例

课程评论表 class_comment ， 对应的字段名称及含义如下：

字段	comment_id	class_id	comment_text	comment_time	stu_id
含义	课程评论ID	课程ID	评论内容	评论时间	学生ID

学生表 student ， 对应的字段名称及含义如下：

字段	stu_id	stu_name	create_time
含义	学生ID	学生昵称	注册时间

在实际应用中，我们在显示课程评论的时候，通常会显示这个学生的昵称，而不是学生 ID，因此当我们想要查询某个课程的前 1000 条评论时，需要关联 class_comment 和 student这两张表来进行查询。

```
# 查询课程 ID 为 10001 的前 1000 条评论
SELECT p.comment_text, p.comment_time, stu.stu_name
FROM class_comment AS p LEFT JOIN student AS stu
ON p.stu_id = stu.stu_id
WHERE p.class_id = 10001
ORDER BY p.comment_id DESC
LIMIT 1000;
```

我们为学生表和课程评论表随机模拟出百万量级的数据，执行关联查询运行时长为接近0.5秒，对于网站的响应来说，这已经很慢了，用户体验会非常差。如果我们想要提升查询的效率，可以允许适当的数据冗余，也就是在商品评论表中增加用户昵称字段， 在class_comment 数据表的基础上增加 stu_name 字段。 这样一来，只需单表查询就可以得到数据集结果：

```
SELECT comment_text, comment_time, stu_name
FROM class_comment
WHERE class_id = 10001
ORDER BY class_id DESC LIMIT 1000;
```

优化之后只需要扫描一次聚集索引即可，查询时间约为之前的 1/10。 在数据量大的情况下，牺牲部分空间减少关联查询，查询效率会有显著的提升。

(3) 相关问题

反范式的新问题：

- 存储空间变大了
- 一个表中字段做了修改，另一个表中冗余的字段也需要做同步修改，否则 数据不一致
- 若采用存储过程来支持数据的更新、删除等额外操作，如果更新频繁，会非常消耗系统资源
- 在数据量小的情况下，反范式不能体现性能的优势，可能还会让数据库的设计更加复杂

使用场景：

只有当**冗余信息有价值或者能大幅度提高查询效率**的时候，我们才会采取反范式的优化。 例如：比如订单中的收货人信息，包括姓名、电话和地址等。每次发生的订单收货信息都属于历史快照，需要进行保存，但用户可以随时修改自己的信息，这时保存这些冗余信息是非常有必要的。

五.全文概览