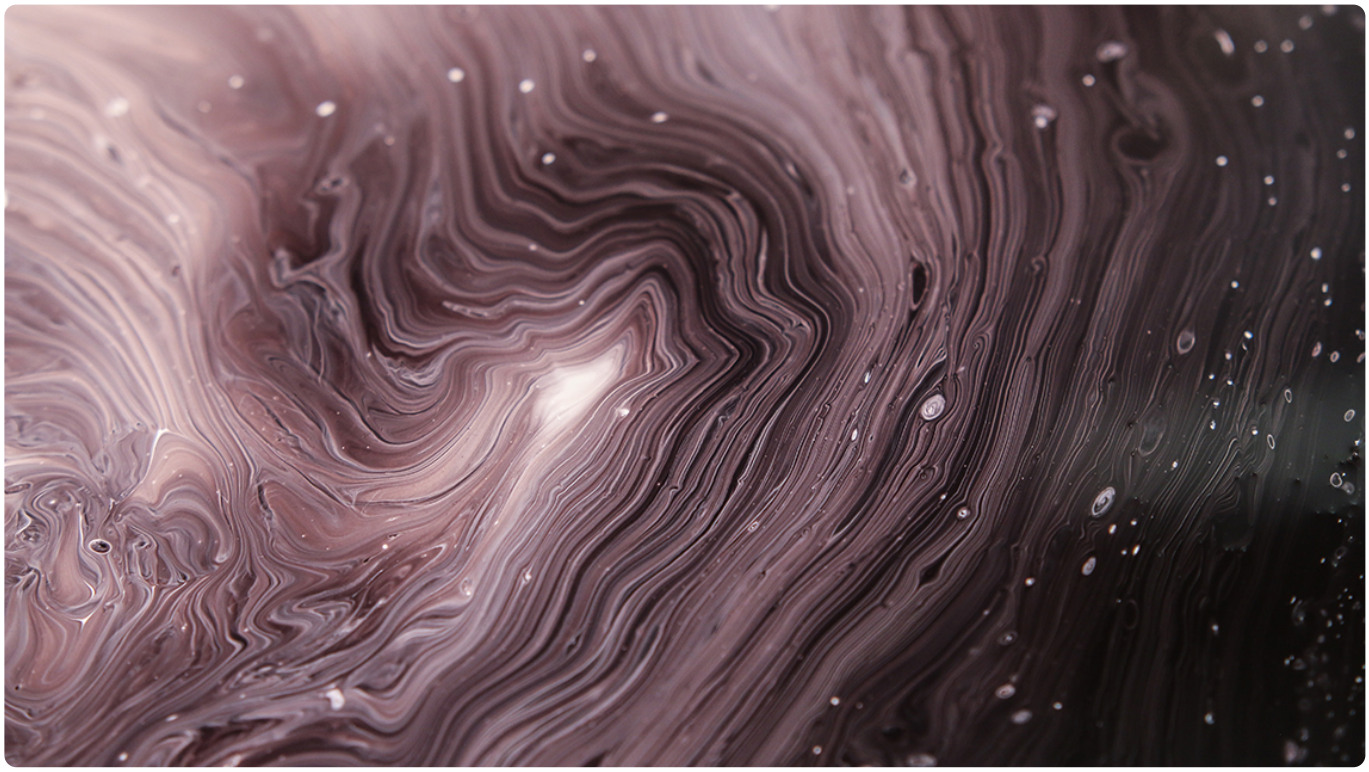
22 | 反范式设计：3NF有什么不足，为什么有时候需要反范式设计？

2019-07-31 陈旸

SQL必知必会 进入课程 >



讲述：陈旸
时长 09:26　大小 8.65M

上一篇文章中，我们介绍了数据表设计的三种范式。作为数据库的设计人员，理解范式的设计以及反范式优化是非常有必要的。

为什么这么说呢？了解以下几个方面的内容之后你就明白了。

1. 3NF 有什么不足？除了 3NF，我们为什么还需要 BCNF？
2. 有了范式设计，为什么有时候需要进行反范式设计？
3. 反范式设计适用的场景是什么？又可能存在哪些问题？

BCNF（巴斯范式）

如果数据表的关系模式符合 3NF 的要求，就不存在问题了吗？我们来看下这张仓库管理关系 warehouse_keeper 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

在这个数据表中，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。我们先来梳理下这些属性之间的依赖关系。

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。

这样，我们就可以找到数据表的候选键是（管理员，物品名）和（仓库名，物品名），

然后我们从候选键中选择一个作为主键，比如（仓库名，物品名）。

在这里，主属性是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。非主属性是数量这个属性。

如何判断一张表的范式呢？我们需要根据范式的等级，从低到高来进行判断。

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，因此，数据表符合 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。

既然数据表已经符合了 3NF 的要求，是不是就不存在问题了呢？我们来看下下面的情况：

1. 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. 如果仓库更换了管理员，我们就可能会修改数据表中的多条记录；
3. 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。

你能看到，即便数据表符合 3NF 的要求，同样可能存在插入，更新和删除数据的异常情况。

这种情况下该怎么解决呢？

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。人们在 3NF 的基础上进行了改进，提出了**BCNF，也叫做巴斯 - 科德范式，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。**

根据 BCNF 的要求，我们需要把仓库管理关系 warehouse_keeper 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。

反范式设计

尽管围绕着数据表的设计有很多范式，但事实上，我们在设计数据表的时候却不一定要参照这些标准。

我们在之前已经了解了越高阶的范式得到的数据表越多，数据冗余度越低。但有时候，我们在设计数据表的时候，还需要为了性能和读取效率违反范式化的原则。反范式就是相对范式化而言的，换句话说，就是允许少量的冗余，通过空间来换时间。

如果我们想对查询效率进行优化，有时候反范式优化也是一种优化思路。

比如我们想要查询某个商品的前 1000 条评论，会涉及到两张表。

商品评论表 product_comment，对应的字段名称及含义如下：

字段	comment_id	product_id	comment_text	comment_time	user_id
含义	商品评论ID	商品ID	评论内容	评论时间	用户ID

用户表 user，对应的字段名称及含义如下：

字段	user_id	user_name	create_time
含义	用户ID	用户昵称	注册时间

下面，我们就用这两张表模拟一下反范式优化。

实验数据：模拟两张百万量级的数据表

为了更好地进行 SQL 优化实验，我们需要给用户表和商品评论表随机模拟出百万量级的数据。我们可以通过存储过程来实现模拟数据。

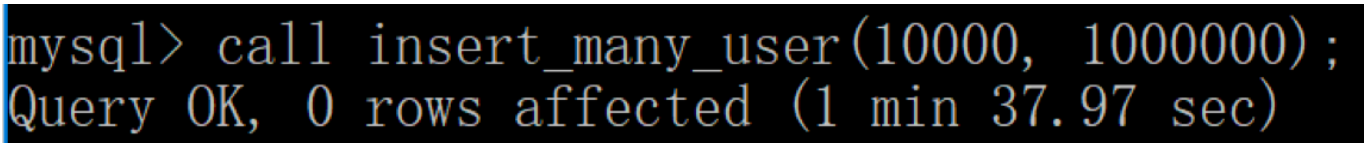
下面是给用户表随机生成 100 万用户的代码：

复制代码

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `insert_many_user`(IN start INT(10), IN max_
BEGIN
DECLARE i INT DEFAULT 0;
DECLARE date_start DATETIME DEFAULT ('2017-01-01 00:00:00');
DECLARE date_temp DATETIME;
SET date_temp = date_start;
SET autocommit=0;
REPEAT
SET i=i+1;
SET date_temp = date_add(date_temp, interval RAND()*60 second);
INSERT INTO user(user_id, user_name, create_time)
VALUES((start+i), CONCAT('user_',i), date_temp);
UNTIL i = max_num
END REPEAT;
COMMIT;
END
```

我用 date_start 变量来定义初始的注册时间，时间为 2017 年 1 月 1 日 0 点 0 分 0 秒，然后用 date_temp 变量计算每个用户的注册时间，新的注册用户与上一个用户注册的时间间隔为 60 秒内的随机值。然后使用 REPEAT ... UNTIL ... END REPEAT 循环，对 max_num 个用户的数据进行计算。在循环前，我们将 autocommit 设置为 0，这样等计算完成再统一插入，执行效率更高。

然后我们来运行 call insert_many_user(10000, 1000000); 调用存储过程。这里需要通过 start 和 max_num 两个参数对初始的 user_id 和要创建的用户数量进行设置。运行结果：



你能看到在 MySQL 里，创建 100 万的用户数据用时 1 分 37 秒。

接着我们再来给商品评论表 product_comment 随机生成 100 万条商品评论。这里我们设置为给某一款商品评论，比如 product_id=10001。评论的内容为随机的 20 个字母。以下是创建随机的 100 万条商品评论的存储过程：

复制代码

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `insert_many_product_comments`(IN START INT
BEGIN
DECLARE i INT DEFAULT 0;
DECLARE date_start DATETIME DEFAULT ('2018-01-01 00:00:00');
DECLARE date_temp DATETIME;
DECLARE comment_text VARCHAR(25);
DECLARE user_id INT;
SET date_temp = date_start;
SET autocommit=0;
REPEAT
SET i=i+1;
SET date_temp = date_add(date_temp, INTERVAL RAND()*60 SECOND);
SET comment_text = substr(MD5(RAND()),1, 20);
SET user_id = FLOOR(RAND()*1000000);
INSERT INTO product_comment(comment_id, product_id, comment_text, comment_time, user_id
VALUES((START+i), 10001, comment_text, date_temp, user_id);
UNTIL i = max_num
END REPEAT;
COMMIT;
END
```

同样的，我用 date_start 变量来定义初始的评论时间。这里新的评论时间与上一个评论的时间间隔还是 60 秒内的随机值，商品评论表中的 user_id 为随机值。我们使用 REPEAT ... UNTIL ... END REPEAT 循环，来对 max_num 个商品评论的数据进行计算。

然后调用存储过程，运行结果如下：

```
mysql> call insert_many_product_comments(10000,1000000);
Query OK, 0 rows affected (2 min 6.98 sec)
```

MySQL 一共花了 2 分 7 秒完成了商品评论数据的创建。

反范式优化实验对比

如果我们想要查询某个商品 ID，比如 10001 的前 1000 条评论，需要写成下面这样：

复制代码

```
SELECT p.comment_text, p.comment_time, u.user_name FROM product_comment AS p
LEFT JOIN user AS u
ON p.user_id = u.user_id
WHERE p.product_id = 10001
ORDER BY p.comment_id DESC LIMIT 1000
```

运行结果（1000 条数据行）：

comment_text	comment_time	user_name
462eed7ac6e791292a79	2018-12-14 04:53:25	user_546655
56910cefd01f6d80f0c7	2018-12-14 04:52:35	user_50353
……	……	……
52f6a51769daf701bc68	2018-12-13 20:35:28	user_698675

运行时长为 0.395 秒，查询效率并不高。

这是因为在实际生活中，我们在显示商品评论的时候，通常会显示这个用户的昵称，而不是用户 ID，因此我们还需要关联 product_comment 和 user 这两张表来进行查询。当表数据量不大的时候，查询效率还好，但如果表数据量都超过了百万量级，查询效率就会变低。这是因为查询会在 product_comment 表和 user 表这两个表上进行聚集索引扫描，然后再嵌套循环，这样一来查询所耗费的时间就有几百毫秒甚至更多。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 product_comment 数据表的基础上增加 user_name 字段，就得到了 product_comment2 数据表。

你可以在百度网盘中下载这三张数据表 product_comment、product_comment2 和 user 表，密码为 n3l8。

这样一来，只需单表查询就可以得到数据集结果：

复制代码

```
SELECT comment_text, comment_time, user_name FROM product_comment2 WHERE product_id = 1(
```

运行结果（1000 条数据）：

comment_text	comment_time	user_name
462eed7ac6e791292a79	2018-12-14 04:53:25	user_546655
56910cefd01f6d80f0c7	2018-12-14 04:52:35	user_50353
.....	.....	.....
52f6a51769daf701bc68	2018-12-13 20:35:28	user_698675

优化之后只需要扫描一次聚集索引即可，运行时间为 0.039 秒，查询时间是之前的 1/10。你能看到，在数据量大的情况下，查询效率会有显著的提升。

反范式存在的问题 & 适用场景

从上面的例子中可以看出，反范式可以通过空间换时间，提升查询的效率，但是反范式也会带来一些新问题。

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。比如采用存储过程来支持数据的更新、删除等额外操作，很容易增加系统的维护成本。

比如用户每次更改昵称的时候，都需要执行存储过程来更新，如果昵称更改频繁，会非常消耗系统资源。

那么反范式优化适用于哪些场景呢？

在现实生活中，我们经常需要一些冗余信息，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。

当冗余信息有价值或者能大幅度提高查询效率的时候，我们就可以采取反范式的优化。

此外反范式优化也常用在数据仓库的设计中，因为数据仓库通常存储历史数据，对增删改的实时性要求不强，对历史数据的分析需求强。这时适当允许数据的冗余度，更方便进行数据分析。

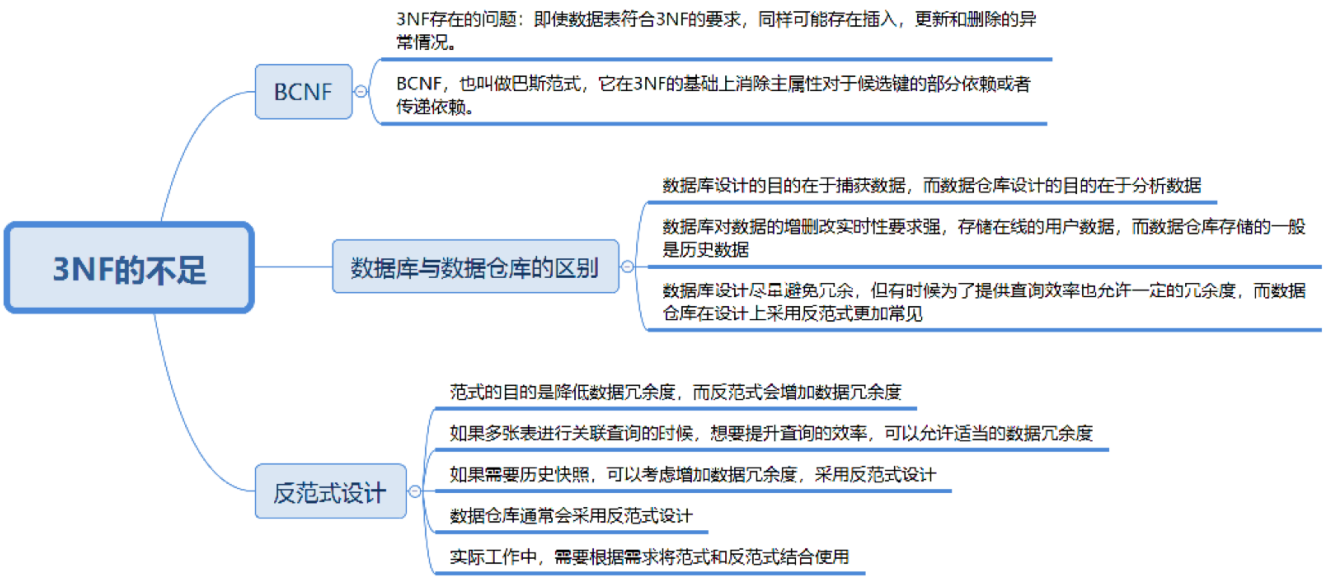
我简单总结下数据仓库和数据库在使用上的区别：

1. 数据库设计的目的在于捕获数据，而数据仓库设计的目的在于分析数据；
2. 数据库对数据的增删改实时性要求强，需要存储在线的用户数据，而数据仓库存储的一般是历史数据；
3. 数据库设计需要尽量避免冗余，但为了提高查询效率也允许一定的冗余度，而数据仓库在设计上更偏向采用反范式设计。

总结

今天我们讲了 BCNF，它是基于 3NF 进行的改进。你能看到设计范式越高阶，数据表就会越精细，数据的冗余度也就越少，在一定程度上可以让数据库在内部关联上更好地组织数据。但有时候我们也需要采用反范进行优化，通过空间来换取时间。

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，我们在数据表的设计中，还需要根据需求将范式和反范式混合使用。



我们今天举了一个反范式设计的例子，你在工作中是否有做过反范式设计的例子？欢迎你在评论区与我们一起分享，也欢迎把这篇文章分享给你的朋友或者同事，一起交流一下。