29丨为什么没有理想的索引?

我之前讲过页这个结构，表和索引都会存储在页中，不同的 DBMS 默认的页的大小是不同的，同时我们也了解到 DBMS 会有缓冲池的机制，在缓冲池里需要有足够多的空间，存储经常被使用到的页，尽可能减少直接的磁盘 I/O 操作。这种策略对 SQL 查询的底层执行来说非常重要，可以从物理层面上最大程度提升 SQL 的查询效率。

但同时我们还需要关注索引的设计，如果只是针对 SQL 查询，我们是可以设计出理想的索引的，不过在实际工作中这种理想的索引往往会带来更多的资源消耗。这是为什么呢？今天我们就来对这部分内容进行学习，内容包括以下几个部分：

1. 什么是索引片？如何计算过滤因子?
2. 设计索引的时候，可以遵循哪些原则呢?
3. 为什么理想的索引很难在实际工作中应用起来?

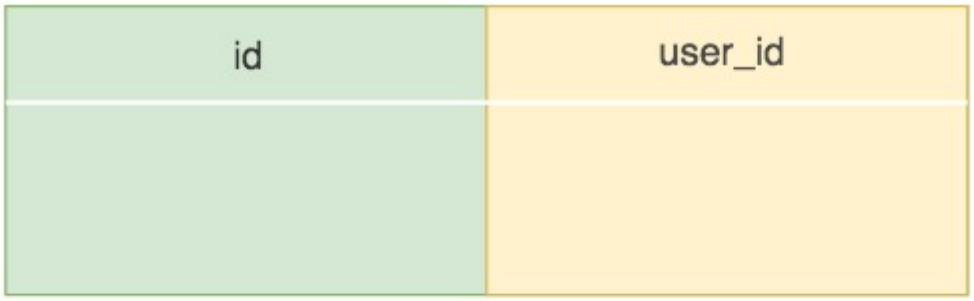
索引片和过滤因子

索引片就是 SQL 查询语句在执行中需要扫描的一个索引片段，我们会根据索引片中包含的匹配列的数量不同，将索引分成窄索引（比如包含索引列数为 1 或 2）和宽索引（包含的索引列数大于 2）。

如果索引片越宽，那么需要顺序扫描的索引页就越多；如果索引片越窄，就会减少索引访问的开销。比如在 product_comment 数据表中，我们将 comment_id 设置为主键，然后执行下面的 SQL 查询语句：

复制代码

```
1 SELECT comment_id, product_id, comment_text, user_id FROM product_comment WHERE user_id
```



窄索引

宽索引

针对这条 SQL 查询语句，我们可以设置窄索引（user_id）。需要说明的是，每个非聚集索引保存的数据都会存储主键值，然后通过主键值，来回表查找相应的数据，因此每个索引都相当于包括了主键，也就是（comment_id, user_id）。

同样我们可以设置宽索引（user_id, product_id, comment_text），相当于包括了主键，也就是（comment_id, user_id, product_id, comment_text）。

如何通过宽索引避免回表

刚才我讲到了宽索引需要顺序扫描的索引页很多，不过它也可以避免通过索引找到主键，再通过主键回表进行数据查找的情况。回表指的就是数据库根据索引找到了数据行之后，还需要通过主键再次到数据表中读取数据的情况。

我们可以用不同索引片来运行下刚才的 SQL 语句，比如我们采用窄索引（user_id）的方式，来执行下面这条语句：

复制代码

```
1 SELECT comment_id, product_id, comment_text, user_id FROM product_comment WHERE user_id
```

运行结果（110 条记录，运行时间 0.062s）：

comment_id	product_id	comment_text	user_id
295895	10001	0a74ba347ffa4b098ba5	100001
995095	10001	1f4d398463aea129ad62	100001
.....	.....	.....	.....
643842	10001	f36f234a6f5f4063b837	100098

同样，如果我们设置宽索引（`user_id, product_id, comment_text`），然后执行相同的 SQL 语句，运行结果相同，运行时间为 0.043s，你能看到查询效率有了一些提升。这就是因为我们可以通过宽索引将 SELECT 中需要用到的列（主键列可以除外）都设置在宽索引中，这样就避免了回表扫描的情况，从而提升 SQL 查询效率。

什么是过滤因子

在索引片的设计中，我们还需要考虑一个因素，那就是过滤因子，它描述了谓词的选择性。在 WHERE 条件语句中，每个条件都称为一个谓词，谓词的选择性也等于满足这个条件列的记录数除以总记录数的比例。

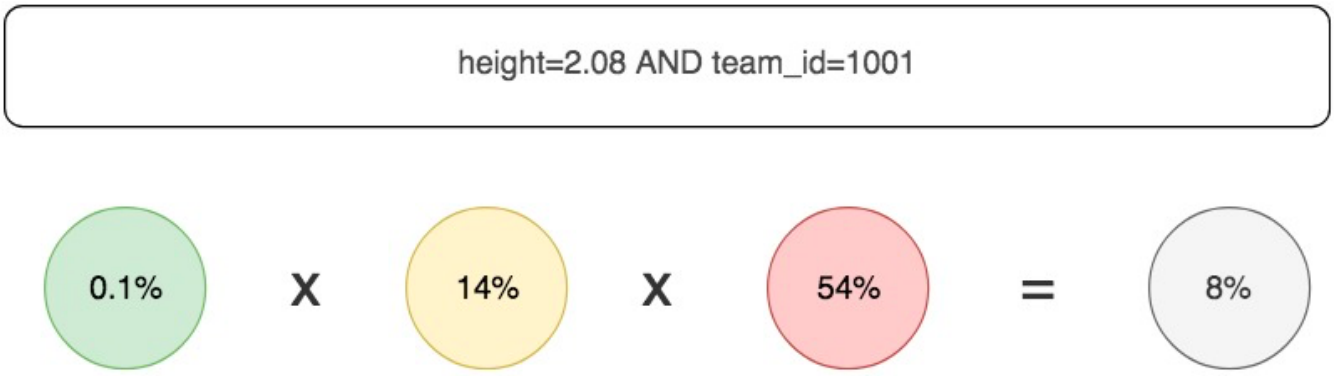
举个例子，我们在 player 数据表中，定义了 team_id 和 height 字段，我们也可以设计个 gender 字段，这里 gender 的取值都为 male。

在 player 表中记录比较少，一共 37 条记录，不过我们也可以统计以下字段：gender、team_id、height 和 name，以便评估过滤因子的筛选能力，如下表所示：

你能看到`gender='male'`不是个好过滤因子，因为所有球员都是男性，同样`team_id=10`

01也不是个好过滤因子，因为这个比例在这个特定的数据集中高达 54%，相比之下height=2.08具有一定的筛选性，过滤因子能力最强的是 name 字段。

这时如果我们创建一个联合的过滤条件（height, team_id），那么它的过滤能力是怎样的呢？



联合过滤因子有更高的过滤能力，这里还需要注意一个条件，那就是条件列的关联性应该尽量相互独立，否则如果列与列之间具有相关性，联合过滤因子的能力就会下降很多。比如城市名称和电话区号就有强相关性，这两个列组合到一起不会加强过滤效果。

你能看到过滤因子决定了索引片的大小（注意这里不是窄索引和宽索引），过滤因子的条件过滤能力越强，满足条件的记录数就越少，SQL 查询需要扫描的索引片也就越小。同理，如果我们没有选择好索引片中的过滤因子，就会造成索引片中的记录数过多的情况。

针对 SQL 查询的理想索引设计：三星索引

刚才我介绍了宽索引和窄索引，有些时候宽索引可以提升 SQL 的查询效率，那么你可能会问，如果针对 SQL 查询来说，有没有一个标准能让 SQL 查询效率最大化呢？

实际上，存在着一个三星索引的标准，这就好比我们在学习数据表设计时提到的三范式一样。三星索引具体指的是：

1. 在 WHERE 条件语句中，找到所有等值谓词中的条件列，将它们作为索引片中的开始列；
2. 将 GROUP BY 和 ORDER BY 中的列加入到索引中；
3. 将 SELECT 字段中剩余的列加入到索引片中。

你能看到这样操作下来，索引片基本上会变成一个宽索引，把能添加的相关列都加入其中。为什么对于一条 SQL 查询来说，这样做的效率是最高的吗？

首先，如果我们要通过索引查找符合条件的记录，就需要将 WHERE 子句中的等值谓词列加入到索引片中，这样索引的过滤能力越强，最终扫描的数据行就越少。

另外，如果我们要对数据记录分组或者排序，都需要重新扫描数据记录。为了避免进行 filesort 排序，可以把 GROUP BY 和 ORDER BY 中涉及到的列加入到索引中，因为创建了索引就会按照索引的顺序来存储数据，这样再对这些数据按照某个字段进行分组或者排序的时候，就会提升效率。

最后，我们取数据的时候，可能会存在回表情况。回表就是通过索引找到了数据行，但是还需要通过主键的方式在数据表中查找完成的记录。这是因为 SELECT 所需的字段并不都保存在索引中，因此我们可以将 SELECT 中的字段都保存在索引中避免回表的情况，从而提升查询效率。

为什么很难存在理想的索引设计

从三星索引的创建过程中，你能看到三星索引实际上分析了在 SQL 查询过程中所有可能影响效率的环节，通过在索引片中添加索引的方式来提升效率。通过上面的原则，我们可以很

快创建一个 SQL 查询语句的三星索引（有时候可能只有两星，比如同时拥有范围谓词和 ORDER BY 的时候）。

但就同三范式一样，很多时候我们并没有遵循三范式的设计原则，而是采用了反范式设计。同样，有时候我们并不能需要完全遵循三星索引的原则，原因主要有以下两点：

1. 采用三星索引会让索引片变宽，这样每个页能够存储的索引数据就会变少，从而增加了页加载的数量。从另一个角度来看，如果数据量很大，比如有 1000 万行数据，过多索引所需要的磁盘空间可能会成为一个问题，对缓冲池所需空间的压力也会增加。
2. 增加了索引维护的成本。如果我们为所有的查询语句都设计理想的三星索引，就会让数据表中的索引个数过多，这样索引维护的成本也会增加。举个例子，当我们添加一条记录的时候，就需要在每一个索引上都添加相应的行（存储对应的主键值），假设添加一行记录的时间成本是 10ms（磁盘随机读取一个页的时间），那么如果我们创建了 10 个索引，添加一条记录的时间就可能变成 0.1s，如果是添加 10 条记录呢？就会花费近 1s 的时间。从索引维护的成本来看消耗还是很高的。当然对于数据库来说，数据的更新不一定马上回写到磁盘上，但即使不及时将脏页进行回写，也会造成缓冲池中的空间占用过多，脏页过多的情况。

总结

你能看到针对一条 SQL 查询来说，三星索引是个理想的方式，但实际运行起来我们要考虑更多维护的成本，在索引效率和索引维护之间进行权衡。

三星索引会让索引变宽，好处就是不需要进行回表查询，减少了磁盘 I/O 的次数，弊端就是会造成频繁的页分裂和页合并，对于数据的插入和更新来说，效率会降低不少。

那我们该如何设计索引呢？

首先一张表的索引个数不宜过多，否则一条记录的增加和修改，会因为过多的索引造成额外的负担。针对这个情况，当你需要新建索引的时候，首先考虑在原有的索引片上增加索引，也就是采用复合索引的方式，而不是新建一个新的索引。另外我们可以定期检查索引的使用情况，对于很少使用到的索引可以及时删除，从而减少索引数量。

同时，在索引片中，我们也需要控制索引列的数量，通常情况下我们将 WHERE 里的条件列添加到索引中，而 SELECT 中的非条件列则不需要添加。除非 SELECT 中的非条件列数少，并且该字段会经常使用到。

另外单列索引和复合索引的长度也需要控制，在 MySQL InnoDB 中，系统默认单个索引长度最大为 767 bytes，如果单列索引长度超过了这个限制，就会取前缀索引，也就是取前 255 字符。这实际上也是告诉我们，字符列会占用较大的空间，在数据表设计的时候，尽量采用数值类型替代字符类型，尽量避免用字符类型做主键，同时针对字符字段最好只建前缀索引。

给你留一道思考题吧，针对下面的 SQL 语句，如果创建三星索引该如何创建？使用三星索引和不使用三星索引在查询效率上又有什么区别呢？

复制代码

```
1 SELECT comment_id, comment_text, user_id FROM product_comment where user_id BETWEEN 100(
```