26丨索引的使用原则：如何通过索引让SQL查询效率最大化？

2019-08-09 陈旸

SQL必知必会 进入课程 >

讲述：陈旸
时长 12:20 大小 11.30M

我之前讲了索引的使用和它的底层原理，今天我来讲一讲索引的使用原则。既然我们的目标是提升 SQL 的查询效率，那么该如何通过索引让效率最大化呢？

今天的课程主要包括下面几个部分：

1. 什么情况下使用索引？当我们进行数据表查询的时候，都有哪些特征需要我们创建索引？
2. 索引不是万能的，索引设计的不合理可能会阻碍数据库和业务处理的性能。那么什么情况下不需要创建索引？
3. 创建了索引不一定代表一定用得上，甚至在有些情况下索引会失效。哪些情况下，索引会失效呢？又该如何避免这一情况？

创建索引有哪些规律？

创建索引有一定的规律。当这些规律出现的时候，我们就可以通过创建索引提升查询效率，下面我们来看看什么情况下可以创建索引：

1. 字段的数值有唯一性的限制，比如用户名

索引本身可以起到约束的作用，比如唯一索引、主键索引都是可以起到唯一性约束的，因此在我们的数据表中，如果某个字段是唯一性的，就可以直接创建唯一性索引，或者主键索引。

2. 频繁作为 WHERE 查询条件的字段，尤其在数据表大的情况下

在数据量大的情况下，某个字段在 SQL 查询的 WHERE 条件中经常被使用到，那么就需要给这个字段创建索引了。创建普通索引就可以大幅提升数据查询的效率。

我之前列举了 product_comment 数据表，这张数据表中一共有 100 万条数据，假设我们想要查询 user_id=785110 的用户对商品的评论。

如果我们没有对 user_id 字段创建索引，进行如下查询：

复制代码

```
SELECT comment_id, product_id, comment_text, comment_time, user_id FROM product_comment
```

运行结果：

comment_id	product_id	comment_text	comment_time	user_id
900002	10001	cb672c3ff4c4f79ef0c6	2018-11-06 01:11:17	785110

运行时间为 0.699s，你能看到查询效率还是比较低的。当我们对 user_id 字段创建索引之后，运行时间为 0.047s，不到原来查询时间的 1/10，效率提升还是明显的。

3. 需要经常 GROUP BY 和 ORDER BY 的列

索引就是让数据按照某种顺序进行存储或检索，因此当我们使用 GROUP BY 对数据进行分组查询，或者使用 ORDER BY 对数据进行排序的时候，就需要对分组或者排序的字段进行索引。

比如我们按照 user_id 对商品评论数据进行分组，显示不同的 user_id 和商品评论的数量，显示 100 个即可。

如果我们不对 user_id 创建索引，执行下面的 SQL 语句：

复制代码

```
SELECT user_id, count(*) as num FROM product_comment group by user_id limit 100
```

运行结果（100 条记录，运行时间 1.666s ）：

user_id	num
912178	2
714098	2
.....	.....
333479	2

如果我们对 user_id 创建索引，再执行 SQL 语句：

复制代码

```
SELECT user_id, count(*) as num FROM product_comment group by user_id limit 100
```

运行结果（100 条记录，运行时间 0.042s ）：

user_id	num
2	2
9	4
.....	.....
174	2

你能看到当对 user_id 创建索引后，得到的结果中 user_id 字段的数值也是按照顺序展示的，运行时间却不到原来时间的 1/40，效率提升很明显。

同样，如果是 ORDER BY，也需要对字段创建索引。我们再来看下同时有 GROUP BY 和 ORDER BY 的情况。比如我们按照 user_id 进行评论分组，同时按照评论时间降序的方式进行排序，这时我们就需要同时进行 GROUP BY 和 ORDER BY，那么是不是需要单独创建 user_id 的索引和 comment_time 的索引呢？

当我们对 user_id 和 comment_time 分别创建索引，执行下面的 SQL 查询：

复制代码

```
1 SELECT user_id, count(*) as num FROM product_comment group by user_id order by comment_t
```

运行结果（运行时间 >100s）：

User_id	num
556655	1
60353	1
.....	.....
755628	1

实际上多个单列索引在多条件查询时只会生效一个索引（MySQL 会选择其中一个限制最严格的作为索引），所以在多条件联合查询的时候最好创建联合索引。在这个例子中，我们创建联合索引 (user_id, comment_time)，再来看下查询的时间，查询时间为 0.775s，效率提升了很多。如果我们创建联合索引的顺序为 (comment_time, user_id) 呢？运行时间为 1.990s，同样比两个单列索引要快，但是会比顺序为 (user_id, comment_time) 的索引要慢一些。这是因为在进行 SELECT 查询的时候，先进行 GROUP BY，再对数据进行 ORDER BY 的操作，所以按照这个联合索引的顺序效率是最高的。

索引	运行时间
两个单索引：user_id, comment_time	>100s
联合索引：(user_id, comment_time)	0.775s
联合索引：(comment_time, user_id)	1.990s

4.UPDATE、DELETE 的 WHERE 条件列，一般也需要创建索引

我们刚才说的是数据检索的情况。那么当我们对某条数据进行 UPDATE 或者 DELETE 操作的时候，是否也需要对 WHERE 的条件列创建索引呢？

我们先看一下对数据进行 UPDATE 的情况。

如果我们想要把 comment_text 为 462eed7ac6e791292a79 对应的 product_id 修改为 10002，当我们没有对 comment_text 进行索引的时候，执行 SQL 语句：

复制代码

```
1 UPDATE product_comment SET product_id = 10002 WHERE comment_text = '462eed7ac6e791292a7
```

运行结果为 Affected rows: 1，运行时间为 1.173s。

你能看到效率不高，但如果我们对 comment_text 字段创建了索引，然后再把刚才那条记录更新回 product_id=10001，执行 SQL 语句：

复制代码

```
UPDATE product_comment SET product_id = 10001 WHERE comment_text = '462eed7ac6e791292a7
```

运行结果为 Affected rows: 1，运行时间仅为 0.1110s。你能看到这个运行时间是之前的 1/10，效率有了大幅的提升。

如果我们对某条数据进行 DELETE，效率如何呢？

比如我们想删除 comment_text 为 462eed7ac6e791292a79 的数据。当我们没有对 comment_text 字段进行索引的时候，执行 SQL 语句：

复制代码

```
DELETE FROM product_comment WHERE comment_text = '462eed7ac6e791292a79'
```

运行结果为 Affected rows: 1，运行时间为 1.027s，效率不高。

如果我们对 comment_text 创建了索引，再来执行这条 SQL 语句，运行时间为 0.032s，时间是原来的 1/32，效率有了大幅的提升。

你能看到，对数据按照某个条件进行查询后再进行 UPDATE 或 DELETE 的操作，如果对 WHERE 字段创建了索引，就能大幅提升效率。原理是因为我们需要先根据 WHERE 条件列检索出来这条记录，然后再对它进行更新或删除。如果进行更新的时候，更新的字段是非索引字段，提升的效率会更明显，这是因为非索引字段更新不需要对索引进行维护。

不过在实际工作中，我们也需要注意平衡，如果索引太多了，在更新数据的时候，如果涉及到索引更新，就会造成负担。

5.DISTINCT 字段需要创建索引

有时候我们需要对某个字段进行去重，使用 DISTINCT，那么对这个字段创建索引，也会提升查询效率。

比如我们想要查询商品评论表中不同的 user_id 都有哪些，如果我们没有对 user_id 创建索引，执行 SQL 语句，看看情况是怎样的。

复制代码

```
SELECT DISTINCT(user_id) FROM `product_comment`
```

运行结果（600637 条记录，运行时间 2.283s ）：

user_id
912178
714098
.....
556655

如果我们对 user_id 创建索引，再执行 SQL 语句，看看情况又是怎样的。

复制代码

```
SELECT DISTINCT(user_id) FROM `product_comment`
```

运行结果（600637 条记录，运行时间 0.627s ）：

user_id
2
9
.....
999998

你能看到 SQL 查询效率有了提升，同时显示出来的 user_id 还是按照递增的顺序进行展示的。这是因为索引会对数据按照某种顺序进行排序，所以在去重的时候也会快很多。

6. 做多表 JOIN 连接操作时，创建索引需要注意以下的原则

首先，连接表的数量尽量不要超过 3 张，因为每增加一张表就相当于增加了一次嵌套的循环，数量级增长会非常快，严重影响查询的效率。

其次，对 WHERE 条件创建索引，因为 WHERE 才是对数据条件的过滤。如果在数据量非常大的情况下，没有 WHERE 条件过滤是非常可怕的。

最后，对用于连接的字段创建索引，并且该字段在多张表中的类型必须一致。比如 user_id 在 product_comment 表和 user 表中都为 int(11) 类型，而不能一个为 int 另一个为 varchar 类型。

举个例子，如果我们只对 user_id 创建索引，执行 SQL 语句：

复制代码

```
SELECT comment_id, comment_text, product_comment.user_id, user_name FROM product_comment
WHERE comment_text = '462eed7ac6e791292a79'
```

运行结果（1 条数据，运行时间 0.810s）：

comment_id	comment_text	user_id	user_name
1010000	462eed7ac6e791292a79	556655	user_546655

这里我们对 comment_text 创建索引，再执行上面的 SQL 语句，运行时间为 0.046s。

如果我们不使用 WHERE 条件查询，而是直接采用 JOIN...ON...进行连接的话，即使使用了各种优化手段，总的运行时间也会很长（ >100s ）。

什么时候不需要创建索引

我之前讲到过索引不是万能的，有一些情况是不需要创建索引的，这里再进行一下说明。

WHERE 条件（包括 GROUP BY、ORDER BY）里用不到的字段不需要创建索引，索引的价值是快速定位，如果起不到定位的字段通常是不需要创建索引的。举个例子：

复制代码

```
SELECT comment_id, product_id, comment_time FROM product_comment WHERE user_id = 41251
```

因为我们是按照 user_id 来进行检索的，所以不需要对其他字段创建索引，即使这些字段出现在 SELECT 字段中。

第二种情况是，如果表记录太少，比如少于 1000 个，那么是不需要创建索引的。我之前讲过一个 SQL 查询的例子（第 23 篇中的 heros 数据表查询的例子，一共 69 个英雄不用索引也很快），表记录太少，是否创建索引对查询效率的影响并不大。

第三种情况是，字段中如果有大量重复数据，也不用创建索引，比如性别字段。不过我们也需要根据实际情况来做判断，这一点我在之前的文章里已经进行了说明，这里不再赘述。

最后一种情况是，频繁更新的字段不一定要创建索引。因为更新数据的时候，也需要更新索引，如果索引太多，在更新索引的时候也会造成负担，从而影响效率。

什么情况下索引失效

我们创建了索引，还要避免索引失效，你可以先思考下都有哪些情况会造成索引失效呢？下面是一些常见的索引失效的例子：

1. 如果索引进行了表达式计算，则会失效

我们可以使用 EXPLAIN 关键字来查看 MySQL 中一条 SQL 语句的执行计划，比如：

复制代码

```
EXPLAIN SELECT comment_id, user_id, comment_text FROM product_comment WHERE comment_id+1
```

运行结果：

复制代码

```
+----+-------------+-----------------+------------+------+---------------+------+------
| id | select_type | table           | partitions | type | possible_keys | key  | key_le
+----+-------------+-----------------+------------+------+---------------+------+------
|  1 | SIMPLE      | product_comment | NULL       | ALL  | NULL          | NULL | NULL
+----+-------------+-----------------+------------+------+---------------+------+------
```

你能看到如果对索引进行了表达式计算，索引就失效了。这是因为我们需要把索引字段的取值都取出来，然后依次进行表达式的计算来进行条件判断，因此采用的就是全表扫描的方式，运行时间也会慢很多，最终运行时间为 2.538 秒。

为了避免索引失效，我们对 SQL 进行重写：

复制代码

```
SELECT comment_id, user_id, comment_text FROM product_comment WHERE comment_id = 900000
```

运行时间为 0.039 秒。

2. 如果对索引使用函数，也会造成失效

比如我们想要对 comment_text 的前三位为 abc 的内容进行条件筛选，这里我们来查看下执行计划：

复制代码

```
EXPLAIN SELECT comment_id, user_id, comment_text FROM product_comment WHERE SUBSTRING(c
```

运行结果：

复制代码

```
+----+-------------+-----------------+------------+------+---------------+------+------
| id | select_type | table           | partitions | type | possible_keys | key  | key_le
+----+-------------+-----------------+------------+------+---------------+------+------
|  1 | SIMPLE      | product_comment | NULL       | ALL  | NULL          | NULL | NULL
+----+-------------+-----------------+------------+------+---------------+------+------
```

你能看到对索引字段进行函数操作，造成了索引失效，这时可以进行查询重写：

复制代码

```
SELECT comment_id, user_id, comment_text FROM product_comment WHERE comment_text LIKE 'a
```

使用 EXPLAIN 对查询语句进行分析：

复制代码

```
+----+-------------+-----------------+------------+-------+---------------+------------
| id | select_type | table           | partitions | type  | possible_keys | key
+----+-------------+-----------------+------------+-------+---------------+------------
|  1 | SIMPLE      | product_comment | NULL       | range | comment_text  | comment_text
+----+-------------+-----------------+------------+-------+---------------+------------
```

你能看到经过查询重写后，可以使用索引进行范围检索，从而提升查询效率。

3. 在 WHERE 子句中，如果在 OR 前的条件列进行了索引，而在 OR 后的条件列没有进行索引，那么索引会失效。

比如下面的 SQL 语句，comment_id 是主键，而 comment_text 没有进行索引，因为 OR 的含义就是两个只要满足一个即可，因此只有一个条件列进行了索引是没有意义的，只要有条件列没有进行索引，就会进行全表扫描，因此索引的条件列也会失效：

复制代码

```
EXPLAIN SELECT comment_id, user_id, comment_text FROM product_comment WHERE comment_id =
```

运行结果：

复制代码

```
+----+-------------+-----------------+------------+------+---------------+------+------
| id | select_type | table           | partitions | type | possible_keys | key  | key_le
+----+-------------+-----------------+------------+------+---------------+------+------
|  1 | SIMPLE      | product_comment | NULL       | ALL  | PRIMARY       | NULL | NULL
+----+-------------+-----------------+------------+------+---------------+------+------
```

如果我们把 comment_text 创建了索引会是怎样的呢？

复制代码

```
+----+-------------+-----------------+------------+-------------+-----------------------
| id | select_type | table           | partitions | type        | possible_keys
+----+-------------+-----------------+------------+-------------+-----------------------
|  1 | SIMPLE      | product_comment | NULL       | index_merge | PRIMARY,comment_text
+----+-------------+-----------------+------------+-------------+-----------------------
```

你能看到这里使用到了 index merge，简单来说 index merge 就是对 comment_id 和 comment_text 分别进行了扫描，然后将这两个结果集进行了合并。这样做的好处就是避免了全表扫描。

4. 当我们使用 LIKE 进行模糊查询的时候，后面不能是 %

复制代码

```
EXPLAIN SELECT comment_id, user_id, comment_text FROM product_comment WHERE comment_text
```

运行结果：

复制代码

```
+----+-------------+-----------------+------------+------+---------------+------+------
| id | select_type | table           | partitions | type | possible_keys | key  | key_le
+----+-------------+-----------------+------------+------+---------------+------+------
|  1 | SIMPLE      | product_comment | NULL       | ALL  | NULL          | NULL | NULL
+----+-------------+-----------------+------------+------+---------------+------+------
```

这个很好理解，如果一本字典按照字母顺序进行排序，我们会从首位开始进行匹配，而不会对中间位置进行匹配，否则索引就失效了。

5. 索引列与 NULL 或者 NOT NULL 进行判断的时候也会失效。

这是因为索引并不存储空值，所以最好在设计数据表的时候就将字段设置为 NOT NULL 约束，比如你可以将 INT 类型的字段，默认值设置为 0。将字符类型的默认值设置为空字符串 (‘’)。

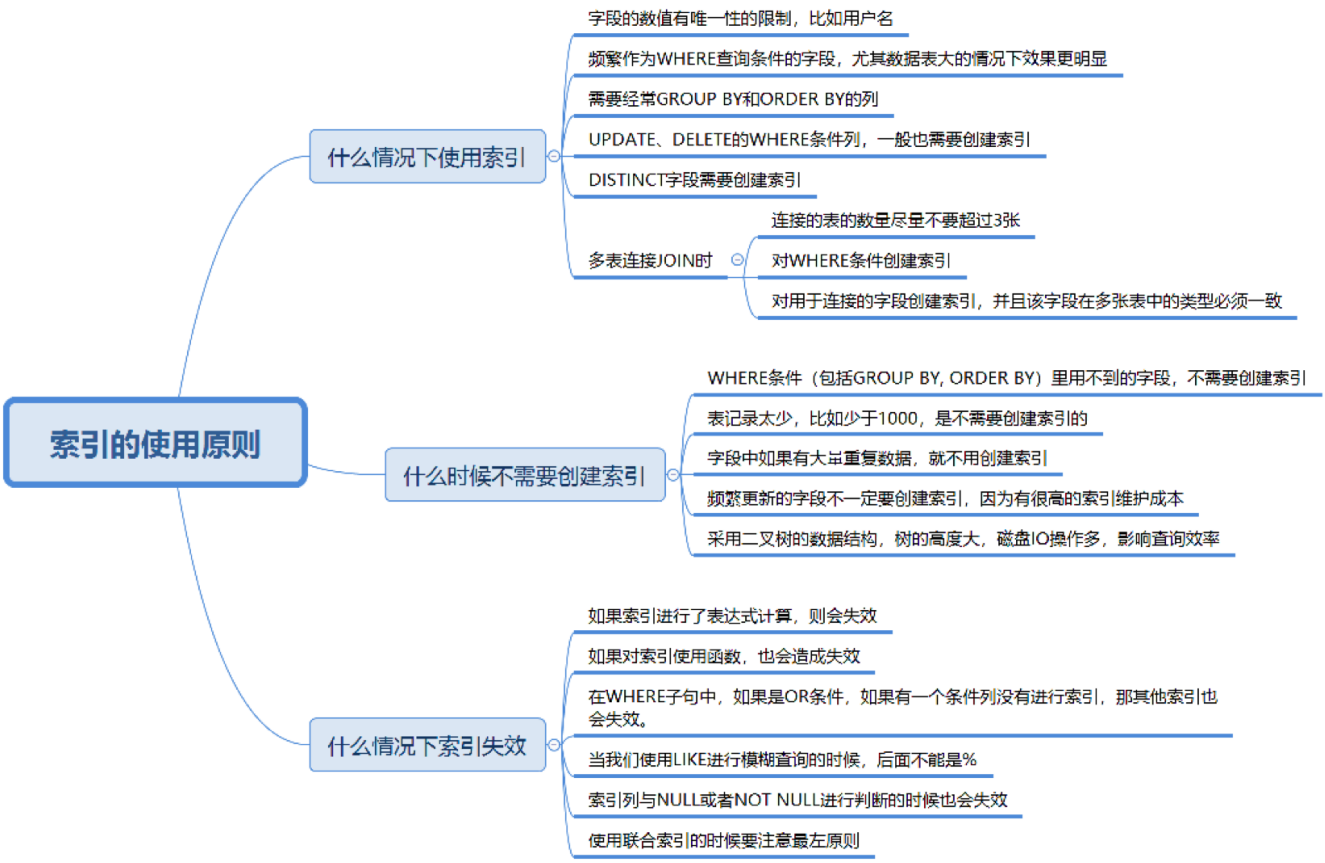
6. 我们在使用联合索引的时候要注意最左原则

最左原则也就是需要从左到右的使用索引中的字段，一条 SQL 语句可以只使用联合索引的一部分，但是需要从最左侧开始，否则就会失效。我在讲联合索引的时候举过索引失效的例子。

总结

今天我们对索引的使用原则进行了梳理，使用好索引可以提升 SQL 查询的效率，但同时 也要注意索引不是万能的。为了避免全表扫描，我们还需要注意有哪些情况可能会导致索引失效，这时就需要进行查询重写，让索引发挥作用。

实际工作中，查询的需求多种多样，创建的索引也会越来越多。这时还需要注意，我们要尽可能扩展索引，而不是新建索引，因为索引数量过多需要维护的成本也会变大，导致写效率变低。同时，我们还需要定期查询使用率低的索引，对于从未使用过的索引可以进行删除，这样才能让索引在 SQL 查询中发挥最大价值。



针对 product_comment 数据表，其中 comment_time 已经创建了普通索引。假设我想查询评论时间在 2018 年 10 月 1 日上午 10 点到 2018 年 10 月 2 日上午 10 点之间的评论，SQL 语句为：

复制代码

```
1 SELECT comment_id, comment_text, comment_time FROM product_comment WHERE DATE(comment_t:
```

你可以想一下这时候索引是否会失效，为什么？如果失效的话，要进行查询重写，应该怎样写？

欢迎你在评论区写下你的答案，也欢迎把这篇文章分享给你的朋友或者同事，一起来交流。

上一篇　25 | Hash索引的底层原理是什么？

下一篇　27 | 从数据页的角度理解B+树查询

精选留言 (15)

写留言

我行我素
2019-08-09

4. 当我们使用 LIKE 进行模糊查询的时候，应该是前面不能是 % 吧

4

haer
2019-08-09

索引失效，因为使用了date函数

展开

2

黑山老妖
2019-08-10

老师 SELECT user_id, count(*) as num FROM product_comment group by user_id order by comment_time desc limit 100
这个例子中 对（comment_time,user_id）进行索引 ，老师不是说按照最左原则，索引会失效嘛 为什么还是会起作用，望老师解答 ： ）

展开

1

梦想天空
2019-08-10

老师 您好。使用selet * from T where a<4 or a=9; a有索引，但还是全盘扫描，不知道什么原因

1

niemo
2019-08-09

老师 您好，sql条件执行顺序不是从右到左么？所有在使用联合索引的时候，把最左的索引写在where条件的最右边，这样理解对么？

展开

1

wusiration
2019-08-09

索引失效，因为使用了date函数。改成SELECT comment_id, comment_text, comment_time FROM product_comment WHERE comment_time BETWEEN DATE('2018-10-01 10:00:00') AND DATE('2018-10-02 10:00:00')

展开

Geek_Wison
2019-08-09

老师您好，本节的内容我有个疑惑的地方：创建联合索引(comment_time, user_id)，但是查询语句是先GROUP BY，然后再ORDER BY，那这样子的话，这个联合索引不是应该不符合最左侧原则而失效了吗？

Ronnyz
2019-08-09

作业：
对comment_time使用了函数，索引失效
SELECT comment_id, comment_text, comment_time FROM product_comment WHE

RE comment_time BETWEEN DATE('2018-10-01 10:00:00') AND DATE('2018-10-02 10:00:00');...

展开

ABC

2019-08-09

索引会失效，因为使用了date函数。

如果修改的话，可以用between和and，对查询条件进行转换。

例如:currtime between date('2018-01-10 10:00:00) and date('2018-02-10 12:00:00...

展开

ttttt

2019-08-09

遇到1055错误，原因是sql_mode=only_full_group_by，不知道有没有遇到的。
执行sql语句：
SELECT user_id, count(*) as num FROM product_comment group by user_id order by comment_time desc limit 100;
报错信息：...

展开

许童童

2019-08-09

会索引失效，用到了DATE函数，应该给字符串使用DATE函数
SELECT comment_id, comment_text, comment_time FROM product_comment WHERE comment_time >= DATE('2018-10-01 10:00:00') AND comment_time <= DATE('2018-10-02 10:00:00')
...

展开

lmtoo

2019-08-09

应该把字符串转为date类型

展开

LiuChen
2019-08-09

老师，对于已有的表结构去创建索引，创建后就直接可以用吗？需不需要其他操作？


蒙开强
2019-08-09

老师，你好，SQL执行where条件的时候是从左到右还是从右到左呢，过滤的时候可以优化先执行过滤数据多的条件。

展开


1


Vackine
2019-08-09

关于关系型数据库模型介绍的论文，老师有推荐的么✨

展开

作者回复: 一本经典的书《Towards a Logical Reconstruction of Relational Database Theory》
另外关系型数据库里面也有不同的使用场景，比如关于图像检索的
《Chabot: Retrieval from a Relational Database of Images》
查询XML的：
《Storing and querying ordered XML using a relational database system》