



下载APP



## 11 | 索引：如何提高查询的速度？

2021-04-01 朱晓峰

MySQL 必知必会

[进入课程 >](#)**讲述：朱晓峰**

时长 12:11 大小 11.16M



你好，我是朱晓峰。

在我们的超市信息系统刚刚开始运营的时候，因为数据量很少，每一次的查询都能很快拿到结果。但是，系统运转时间长了以后，数据量不断地累积，变得越来越庞大，很多查询的速度就变得特别慢。这个时候，我们就采用了 MySQL 提供的高效访问数据的方法——索引，有效地解决了这个问题，甚至之前的一个需要 8 秒钟才能完成的查询，现在只用 0.3 秒就搞定了，速度提升了 20 多倍。

那么，索引到底是啥呢？该怎么使用呢？这节课，我们就来聊一聊。



### 索引是什么？

如果你去过图书馆，应该会知道图书馆的检索系统。图书馆为图书准备了检索目录，包括书名、书号、对应的位置信息，包括在哪个区、哪个书架、哪一层。我们可以通过书名或书号，快速获知书的位置，拿到需要的书。

MySQL 中的索引，就相当于图书馆的检索目录，它是帮助 MySQL 系统快速检索数据的一种存储结构。我们可以在索引中按照查询条件，检索索引字段的值，然后快速定位数据记录的位置，这样就不需要遍历整个数据表了。而且，数据表中的字段越多，表中数据记录越多，速度提升越是明显。

我来举个例子，进一步解释下索引的作用。这里要用到销售流水表（demo.trans），表结构如下：

[复制代码](#)

```
1 mysql> describe demo.trans;
2 +-----+-----+-----+-----+-----+-----+
3 | Field          | Type      | Null  | Key  | Default | Extra |
4 +-----+-----+-----+-----+-----+-----+
5 | itemnumber     | int       | YES   | MUL  | NULL    |      |
6 | quantity       | text      | YES   |      | NULL    |      |
7 | price          | text      | YES   |      | NULL    |      |
8 | transdate      | datetime  | YES   | MUL  | NULL    |      |
9 | actualvalue    | text      | YES   |      | NULL    |      |
10 | barcode        | text      | YES   |      | NULL    |      |
11 | cashiernumber  | int       | YES   | MUL  | NULL    |      |
12 | branchnumber   | int       | YES   | MUL  | NULL    |      |
13 | transuniqueid  | text      | YES   |      | NULL    |      |
14 +-----+-----+-----+-----+-----+-----+
15 9 rows in set (0.02 sec)
```

某个门店的销售流水表有 400 万条数据，现在我要查看一下商品编号是 100 的商品在 2020-12-12 这一天的销售情况，查询代码如下：

[复制代码](#)

```
1 mysql> SELECT
2 -> quantity,price,transdate
3 -> FROM
4 -> demo.trans
5 -> WHERE
6 -> transdate > '2020-12-12'
7 -> AND transdate < '2020-12-13'
8 -> AND itemnumber = 100;
```

```
9  +-----+-----+-----+
10 | quantity | price | transdate |
11 +-----+-----+-----+
12 | 1.000 | 220.00 | 2020-12-12 19:45:36 |
13 | 1.000 | 220.00 | 2020-12-12 08:56:37 |
14 +-----+-----+-----+
15 2 rows in set (8.08 sec)
```

可以看到，结果总共有 2 条记录，可是却花了 8 秒钟，非常慢。同时，这里我没有做表的关联，这只是单表的查询，而且只是一个门店几个月的数据而已。而总部是把所有门店的数据都汇总到一起，查询速度更慢，这样的查询效率，我们肯定是不能接受的。

怎么解决这个问题呢？这时，我们就可以给数据表添加索引。

## 单字段索引

MySQL 支持单字段索引和组合索引，而单字段索引比较常用，我们先来学习下创建单字段索引的方法。

### 如何创建单字段索引？

创建单字段索引，一般有 3 种方式：

1. 你可以通过 CREATE 语句直接给已经存在的表创建索引，这种方式比较简单，我就不多解释了；
2. 可以在创建表的同时创建索引；
3. 可以通过修改表来创建索引。

直接给数据表创建索引的语法如下：

```
1 CREATE INDEX 索引名 ON TABLE 表名 (字段);
```

 复制代码

创建表的同时创建索引的语法如下所示：

 复制代码

```
1 CREATE TABLE 表名
2 (
3  字段 数据类型,
4  ...
5  { INDEX | KEY } 索引名(字段)
6 )
```

修改表时创建索引的语法如下所示：

```
1 ALTER TABLE 表名 ADD { INDEX | KEY } 索引名 (字段);
```

[复制代码](#)

这里有个小问题要提醒你一下，给表设定主键约束或者唯一性约束的时候，MySQL 会自动创建主键索引或唯一性索引。这也是我建议你在创建表的时候，一定要定义主键的原因之一。

举个小例子，我们可以给表 demo.trans 创建索引如下：

```
1 mysql> CREATE INDEX index_trans ON demo.trans (transdate(10));
2 Query OK, 0 rows affected (1 min 8.71 sec)
3 Records: 0 Duplicates: 0 Warnings: 0
4
5 mysql> SELECT
6 -> quantity,price,transdate
7 -> FROM
8 -> demo.trans
9 -> WHERE
10 -> transdate > '2020-12-12'
11 -> AND transdate < '2020-12-13'
12 -> AND itemnumber = 100;
13 +-----+-----+-----+
14 | quantity | price | transdate |
15 +-----+-----+-----+
16 | 1.000 | 220.00 | 2020-12-12 19:45:36 |
17 | 1.000 | 220.00 | 2020-12-12 08:56:37 |
18 +-----+-----+-----+
19 2 rows in set (0.30 sec)
```

[复制代码](#)

可以看到，加了索引之后，这一次我们只用了 0.3 秒，比没有索引的时候，快了 20 多倍。这么大的差距，说明索引对提高查询的速度确实很有帮助。那么，索引是如何做到这一点

的呢？下面我们来学习下单字段索引的作用原理。

## 单字段索引的作用原理

要知道索引是怎么起作用的，我们需要借助 MySQL 中的 EXPLAIN 这个关键字。

EXPLAIN 关键字能够查看 SQL 语句的执行细节，包括表的加载顺序，表是如何连接的，以及索引使用情况等。

 复制代码

```
1 mysql> EXPLAIN SELECT
2 -> quantity,price,transdate
3 -> FROM
4 -> demo.trans
5 -> WHERE
6 -> transdate > '2020-12-12'
7 -> AND transdate < '2020-12-13'
8 -> AND itemnumber = 100;
9 +-----+-----+-----+-----+-----+-----+-----+-----+
10 | id | select_type | table | partitions | type | possible_keys | key | key_len
11 +-----+-----+-----+-----+-----+-----+-----+-----+
12 | 1 | SIMPLE | trans | NULL | range | index_trans | index_trans | 6 | NULL | 5
13 +-----+-----+-----+-----+-----+-----+-----+-----+
14 1 row in set, 1 warning (0.00 sec)
```

我来解释下代码里的关键内容。

type=range：表示使用索引查询特定范围的数据记录。

rows=5411：表示需要读取的记录数。

possible\_keys=index\_trans：表示可以选择的索引是 index\_trans。

key=index\_trans：表示实际选择的索引是 index\_trans。

extra=Using index condition;Using where;Using MRR：这里面的信息对 SQL 语句的执行细节做了进一步的解释，包含了 3 层含义：第一个是执行时使用了索引，第二个是执行时通过 WHERE 条件进行了筛选，第三个是使用了顺序磁盘读取的策略。

通过这个小例子，我们可以发现，有了索引之后，MySQL 在执行 SQL 语句的时候多了一种优化的手段。也就是说，在查询的时候，可以先通过查询索引快速定位，然后再找到对

应的数据进行读取，这样就大大提高了查询的速度。

## 如何选择索引字段？

在刚刚的查询中，我们是选择 transdate（交易时间）字段来当索引字段，你可能会问，为啥不选别的字段呢？这是因为，交易时间是查询条件。MySQL 可以按照交易时间的限定“2020 年 12 月 12 日”，在索引中而不是数据表中寻找满足条件的索引记录，再通过索引记录中的指针来定位数据表中的数据。这样，索引就能发挥作用了。

不过，你有没有想过，itemnumber 字段也是查询条件，能不能用 itemnumber 来创建一个索引呢？我们来试一试：

[复制代码](#)

```
1 mysql> CREATE INDEX index_trans_itemnumber ON demo.trans (itemnumber);
2 Query OK, 0 rows affected (43.88 sec)
3 Records: 0 Duplicates: 0 Warnings: 0
```

然后看看效果：

[复制代码](#)

```
1 mysql> SELECT
2 -> quantity,price,transdate
3 -> FROM
4 -> demo.trans
5 -> WHERE
6 -> transdate > '2020-12-12'      -- 对交易时间的筛选，可以在transdate的索引中定位
7 -> AND transdate < '2020-12-13'
8 -> AND itemnumber = 100;         -- 对商品编号的筛选，可以在itemnumber的索引中定位
9 +-----+-----+-----+
10 | quantity | price | transdate |
11 +-----+-----+-----+
12 | 1.000    | 220.00 | 2020-12-12 19:45:36 |
13 | 1.000    | 220.00 | 2020-12-12 08:56:37 |
14 +-----+-----+-----+
15 2 rows in set (0.38 sec)
```

我们发现，用 itemnumber 创建索引之后，查询速度跟之前差不多，基本在同一个数量级。

这是为啥呢？我们来看看 MySQL 的运行计划：

[复制代码](#)

```
1 mysql> EXPLAIN SELECT
2 -> quantity,price,transdate
3 -> FROM
4 -> demo.trans
5 -> WHERE
6 -> transdate > '2020-12-12'
7 -> AND transdate < '2020-12-13'
8 -> AND itemnumber = 100;                -- 对itemnumber 进行限定
9 +----+-----+-----+-----+-----+-----+-----+-----+
10 | id | select_type | table | partitions | type | possible_keys | key | key_len
11 +----+-----+-----+-----+-----+-----+-----+-----+
12 | 1 | SIMPLE | trans | NULL | ref | index_trans,index_trans_itemnumber | index
13 +----+-----+-----+-----+-----+-----+-----+-----+
14 1 row in set, 1 warning (0.01 sec)
```

我们发现，“possible\_keys= index\_trans,index\_trans\_itemnumber”，就是说 MySQL 认为可以选择的索引确实有 2 个，一个是用 transdate 字段创建的索引 index\_trans，另一个是用 itemnumber 字段创建的索引 index\_trans\_itemnumber。

key= index\_trans\_itemnumber，说明 MySQL 实际选择使用的索引是 itemnumber 字段创建的索引 index\_trans\_itemnumber。而 rows=1192，就表示实际读取的数据记录数只有 1192 个，比用 transdate 创建的索引 index\_trans 的实际读取记录数要少，这就是 MySQL 选择使用 itemnumber 索引的原因。

**所以，我建议你选择索引字段的时候，要选择那些经常被用做筛选条件的字段。**这样才能发挥索引的作用，提升检索的效率。

## 组合索引

在实际工作中，有时会遇到比较复杂的数据表，这种表包括的字段比较多，经常需要通过不同的字段筛选数据，特别是数据表中包含多个层级信息。比如我们的销售流水表就包含了门店信息、收款机信息和商品信息这 3 个层级信息。门店对应多个门店里的收款机，每个收款机对应多个从这台收款机销售出去的商品。我们经常要把这些层次信息作为筛选条件，来进行查询。这个时候单字段的索引往往不容易发挥出索引的最大功效，可以使用组合索引。



现在，先看看单字段索引的效果，我们分别用 branchnumber 和 cashiernumber 来创建索引：

[复制代码](#)

```
1 mysql> CREATE INDEX index_trans_branchnumber ON demo.trans (branchnumber);
2 Query OK, 0 rows affected (41.49 sec)
3 Records: 0 Duplicates: 0 Warnings: 0
4
5 mysql> CREATE INDEX index_trans_cashiernumber ON demo.trans (cashiernumber);
6 Query OK, 0 rows affected (41.95 sec)
7 Records: 0 Duplicates: 0 Warnings: 0
```

有了门店编号和收款机编号的索引，现在我们就尝试一下以门店编号、收款机编号和商品编号为查询条件，来验证一下索引是不是起了作用。

[复制代码](#)

```
1 mysql> SELECT
2 -> itemnumber,quantity,price,transdate
3 -> FROM
4 -> demo.trans
5 -> WHERE
6 -> branchnumber = 11 AND cashiernumber = 1 -- 门店编号和收款机号为筛选条件
7 -> AND itemnumber = 100;    -- 商品编号为筛选条件
8 +-----+-----+-----+-----+
9 | itemnumber | quantity | price | transdate |
10 +-----+-----+-----+-----+
11 | 100 | 1.000 | 220.00 | 2020-07-11 09:18:35 |
12 | 100 | 1.000 | 220.00 | 2020-09-06 21:21:58 |
13 | 100 | 1.000 | 220.00 | 2020-11-10 15:00:11 |
14 | 100 | 1.000 | 220.00 | 2020-12-25 14:28:06 |
15 | 100 | 1.000 | 220.00 | 2021-01-09 20:21:44 |
16 | 100 | 1.000 | 220.00 | 2021-02-08 10:45:05 |
17 +-----+-----+-----+-----+
18 6 rows in set (0.31 sec)
```

结果有 6 条记录，查询时间是 0.31 秒，跟只创建商品编号索引差不多。下面我们就来查看一下执行计划，看看新建的索引有没有起作用。

[复制代码](#)

```
1 mysql> EXPLAIN SELECT
2 -> itemnumber,quantity,price,transdate
3 -> FROM
```



```

4 -> demo.trans
5 -> WHERE
6 -> branchnumber = 11 AND cashiernumber = 1
7 -> AND itemnumber = 100;
8 +----+-----+-----+-----+-----+-----+-----+-----+
9 | id | select_type | table | partitions | type | possible_keys | key | key_len
10 +----+-----+-----+-----+-----+-----+-----+-----+
11 | 1 | SIMPLE | trans | NULL | ref | index_trans_itemnumber,index_trans_branchn
12 +----+-----+-----+-----+-----+-----+-----+-----+
13 1 row in set, 1 warning (0.01 sec)

```

MySQL 有 3 个索引可以用，分别是用 branchnumber 创建的 index\_trans\_branchnumber、用 cashiernumber 创建的 index\_trans\_cashiernumber 和用 itemnumber 创建的 index\_trans\_itemnumber。

最后，MySQL 还是选择了 index\_trans\_itemnumber，实际筛选的记录数是 1192，花费了 0.31 秒。

为什么 MySQL 会这样选呢？这是因为，优化器现在有 3 种索引可以用，分别是商品编号索引、门店编号索引和收款机号索引。优化器发现，商品编号索引实际搜索的记录数最少，所以最后就选择了这种索引。

所以，**如果有多个索引，而这些索引的字段同时作为筛选字段出现在查询中的时候，MySQL 会选择使用最优的索引来执行查询操作。**


能不能让这几个筛选字段同时发挥作用呢？这就用到组合索引了。组合索引，就是包含多个字段的索引。MySQL 最多支持由 16 个字段组成的组合索引。

## 如何创建组合索引？

创建组合索引的语法结构与创建单字段索引相同，不同的是相比单字段索引，组合索引使用了多个字段。

直接给数据表创建索引的语法如下：

```
1 CREATE INDEX 索引名 ON TABLE 表名 (字段1, 字段2, ...);
```

 复制代码

## 创建表的同时创建索引：

[复制代码](#)

```
1 CREATE TABLE 表名
2 (
3  字段 数据类型,
4  ...
5  { INDEX | KEY } 索引名(字段1, 字段2, ...)
6 )
```

## 修改表时创建索引：

[复制代码](#)

```
1 ALTER TABLE 表名 ADD { INDEX | KEY } 索引名 (字段1, 字段2, ...);
```

现在，针对刚刚的查询场景，我们就可以通过创建组合索引，发挥多个字段的筛选作用。

具体做法是，我们给销售流水表创建一个由 3 个字段 branchnumber、cashiernumber、itemnumber 组成的组合索引，如下所示：

[复制代码](#)

```
1 mysql> CREATE INDEX Index_branchnumber_cashiernumber_itemnumber ON demo.trans
2 Query OK, 0 rows affected (59.26 sec)
3 Records: 0 Duplicates: 0 Warnings: 0
```

有了组合索引，刚刚的查询速度就更快了：

[复制代码](#)


```
1 mysql> SELECT
2 -> itemnumber,quantity,price,transdate
3 -> FROM
4 -> demo.trans
5 -> WHERE
6 -> branchnumber = 11 AND cashiernumber = 1
7 -> AND itemnumber = 100;
8 +-----+-----+-----+-----+
9 | itemnumber | quantity | price | transdate |
10 +-----+-----+-----+-----+
11 | 100 | 1.000 | 220.00 | 2020-07-11 09:18:35 |
```

```

12 | 100 | 1.000 | 220.00 | 2020-09-06 21:21:58 |
13 | 100 | 1.000 | 220.00 | 2020-11-10 15:00:11 |
14 | 100 | 1.000 | 220.00 | 2020-12-25 14:28:06 |
15 | 100 | 1.000 | 220.00 | 2021-01-09 20:21:44 |
16 | 100 | 1.000 | 220.00 | 2021-02-08 10:45:05 |
17 +-----+-----+-----+-----+
18 6 rows in set (0.00 sec)

```

几乎是瞬间就完成了，不超过 10 毫秒。我们看看 MySQL 的执行计划：

 复制代码

```

1 mysql> EXPLAIN SELECT
2 -> itemnumber,quantity,price,transdate
3 -> FROM
4 -> demo.trans
5 -> WHERE                                -- 同时筛选门店编号、收款机号和商品编号
6 -> branchnumber = 11 AND cashiernumber = 1
7 -> AND itemnumber = 100;
8 +-----+-----+-----+-----+-----+-----+-----+
9 | id | select_type | table | partitions | type | possible_keys | key | key_len
10 +-----+-----+-----+-----+-----+-----+-----+
11 | 1 | SIMPLE | trans | NULL | ref | index_trans_itemnumber,index_trans_branchn
12 +-----+-----+-----+-----+-----+-----+-----+
13 1 row in set, 1 warning (0.01 sec)

```

这个查询，MySQL 可以用到的索引有 4 个：

index\_trans\_itemnumber;

index\_trans\_branchnumber;

index\_trans\_cashiernumber;

我们刚才用 branchnumber、cashiernumber 和 itemnumber 创建的组合索引  
Index\_branchnumber\_cashiernumber\_itemnumber。

MySQL 选择了组合索引，筛选后读取的记录只有 6 条。组合索引被充分利用，筛选更加精准，所以非常快。

## 组合索引的原理

下面我就来讲讲组合索引的工作原理。

**组合索引的多个字段是有序的，遵循左对齐的原则。**比如我们创建的组合索引，排序的方式是 branchnumber、cashiernumber 和 itemnumber。因此，筛选的条件也要遵循从左向右的原则，如果中断，那么，断点后面的条件就没有办法利用索引了。

比如说我们刚才的条件，branchnumber = 11 AND cashiernumber = 1 AND itemnumber = 100，包含了从左到右的所有字段，所以可以最大限度使用全部组合索引。

假如把条件换成 “cashiernumber = 1 AND itemnumber = 100”，由于我们的组合索引是按照 branchnumber、cashiernumber 和 itemnumber 的顺序建立的，最左边的字段 branchnumber 没有包含到条件当中，中断了，所以这个条件完全不能使用组合索引。

类似的，如果筛选的是一个范围，如果没有办法无法精确定位，也相当于中断。比如 “branchnumber > 10 AND cashiernumber = 1 AND itemnumber = 100” 这个条件，只能用到组合索引中 branchnumber>10 的部分，后面的索引就都用不上了。我们来看看 MySQL 的运行计划：

[复制代码](#)


```
1 mysql> EXPLAIN SELECT
2 -> itemnumber,quantity,price,transdate
3 -> FROM
4 -> demo.trans
5 -> WHERE
6 -> branchnumber > 10 AND cashiernumber = 1 AND itemnumber = 100;
7 +----+-----+-----+-----+-----+-----+-----+-----+
8 | id | select_type | table | partitions | type | possible_keys | key | key_len
9 +----+-----+-----+-----+-----+-----+-----+-----+
10 | 1 | SIMPLE | trans | NULL | ref | index_trans_itemnumber,index_trans_branchn
11 +----+-----+-----+-----+-----+-----+-----+-----+
12 1 row in set, 1 warning (0.02 sec)
```

果然，MySQL 没有选择组合索引，而是选择了用 itemnumber 创建的普通索引 index\_trans\_itemnumber。因为**如果只用组合索引的一部分，效果没有单字段索引那么好。**

## 总结


这节课，我们学习了什么是索引、如何创建和使用索引。索引可以非常显著地提高数据查询的速度，数据表里包含的数据越多，效果越显著。我们应该选择经常被用做筛选条件的字段来创建索引，这样才能通过索引缩小实际读取数据表中数据的范围，发挥出索引的优势。如果有多个筛选的字段，而且经常一起出现，也可以用多个字段来创建组合索引。

如果你要删除索引，就可以用：

 复制代码

```
1 DROP INDEX 索引名 ON 表名;
```

当然，有的索引不能用这种方法删除，比如主键索引，你就必须通过修改表来删除索引。语法如下：

 复制代码

```
1 ALTER TABLE 表名 DROP PRIMARY KEY;
```

最后，我来跟你说说索引的成本。索引能够提升查询的效率，但是建索引也是有成本的，主要有 2 个方面，一个存储空间的开销，还有一个是数据操作上的开销。

存储空间的开销，是指索引需要单独占用存储空间。

数据操作上的开销，是指一旦数据表有变动，无论是插入一条新数据，还是删除一条旧的数据，甚至是修改数据，如果涉及索引字段，都需要对索引本身进行修改，以确保索引能够指向正确的记录。

因此，索引也不是越多越好，创建索引有存储开销和操作开销，需要综合考虑。

## 思考题

假如我有一个单品销售统计表，包括门店编号、销售日期（年月日）、商品编号、销售数量、销售金额、成本、毛利，而用户经常需要对销售情况进行查询，你会对这个表建什么样的索引呢？为什么？

欢迎在留言区写下你的思考和答案，我们一起交流讨论。如果你觉得今天的内容对你有所帮助，也欢迎你分享给你的朋友或同事，我们下节课见。

提建议

12.12 大促

# 每日一课 VIP 年卡

10分钟，解决你的技术难题

¥159/年 ¥365/年

每日一课  
VIP 年卡

仅3天，【点击】图片，立即抢购 >>>

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 10 | 如何进行数学计算、字符串处理和条件判断？

## 精选留言 (6)

写留言



Devo

2021-04-01

我会选择（商品编号，[门店编号]，销售日期）的联合索引，主要依据是商品编号一般是等值查询且区分度较高，门店编号这个字段我觉得可以看具体查询场景选择，销售日期一

一般是范围，放在末位直接扫链表效果较好，请老师指正，谢谢！

展开 ▾



2



**Harry**

2021-04-01

CREATE INDEX index\_trans ON demo.trans (transdate(10));

——括号里面的 10 有什么具体含义吗？

展开 ▾



1



2



**Harry**

2021-04-01

小结部分总结得太好了，从索引的好处、使用方法，到如何选择索引字段，以及索引的使用成本，都覆盖到了。



**lesserror**

2021-04-01

总结一下这一讲的收获吧。

MySQL 最多支持由 16 个字段组成的组合索引。

组合索引的所有组成字段都被查询条件用到，且符合最左匹配原则，查询效率有可能会...

展开 ▾



**Harry**

2021-04-01

在为字段 itemnumber 创建索引 index\_trans\_itemnumber 后，实际读取的记录数下降了 80% (与使用索引 index\_trans 相比)，但为什么查询速度反而下降了 0.08 秒呢？

展开 ▾



**右耳朵猫咪**

2021-04-01



如果执行计划的key是多个单字段索引， 它和一个单字段索引有什么区别呢？

