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

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：

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

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：

```
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：

```
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：

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

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：

```
UPDATE product_comment SET product_id = 10002 WHERE comment_text = '462eed7ac6e791292a79'
```

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：

```
UPDATE product_comment SET product_id = 10001 WHERE comment_text = '462eed7ac6e791292a79'
```

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：

```
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。

```
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。

```
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：

```
SELECT comment_id, comment_text, product_comment.user_id, user_name FROM product_comment JOIN user
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：

```
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：

```
EXPLAIN SELECT comment_id, user_id, comment_text FROM product_comment WHERE comment_id+1 = 900
```

运行结果：

```
+----+-------------+-----------------+------------+------+---------------+------+---------+------+--------+----------+-------------+
| id | select_type | table           | partitions | type | possible_keys | key  | key_len | ref  | rows   | filtered | Extra       |
+----+-------------+-----------------+------------+------+---------------+------+---------+------+--------+----------+-------------+
|  1 | SIMPLE      | product_comment | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 996663 |   100.00 | Using where |
+----+-------------+-----------------+------------+------+---------------+------+---------+------+--------+----------+-------------+
```

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：

```
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：

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

运行结果：

```
+----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------+
| id | select_type | table     | partitions | type | possible_keys | key  | key_len | ref  | rows   | filtered | Extra |
+----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------+
|  1 | SIMPLE      | product_comment | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 996663 |   100.00 |
+----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------+
```

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：

```
SELECT comment_id, user_id, comment_text FROM product_comment WHERE comment_text LIKE 'abc%'
```

使用EXPLAIN对查询语句进行分析：

```
+----+-------------+-----------+------------+-------+---------------+--------------+---------+------+------+----------+-----------------------+
| id | select_type | table     | partitions | type  | possible_keys | key          | key_len | ref  | rows | filtered | Extra                 |
+----+-------------+-----------+------------+-------+---------------+--------------+---------+------+------+----------+-----------------------+
|  1 | SIMPLE      | product_comment | NULL       | range | comment_text  | comment_text | 767     | NULL |  213 |   100.00 | Using index condition |
+----+-------------+-----------+------------+-------+---------------+--------------+---------+------+------+----------+-----------------------+
```

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：

```
EXPLAIN SELECT comment_id, user_id, comment_text FROM product_comment WHERE comment_id = 90000
```

运行结果：

```
+----+-------------+-----------------+------------+------+---------------+------+---------+------+--------+----------+
| id | select_type | table           | partitions | type | possible_keys | key  | key_len | ref  | rows   | filtered | Extra    |
+----+-------------+-----------------+------------+------+---------------+------+---------+------+--------+----------+
|  1 | SIMPLE      | product_comment | NULL       | ALL  | PRIMARY       | NULL | NULL    | NULL | 996663 |    10.00 |
+----+-------------+-----------------+------------+------+---------------+------+---------+------+--------+----------+
```

如果我们把comment_text创建了索引会是怎样的呢？

```
+----+-------------+-----------------+------------+-------------+---------------+---------------+---------+------+------+----------+-----------------------+
| id | select_type | table           | partitions | type        | possible_keys | key           | key_len | ref  | rows | filtered | Extra                 |
+----+-------------+-----------------+------------+-------------+---------------+---------------+---------+------+------+----------+-----------------------+
|  1 | SIMPLE      | product_comment | NULL       | index_merge | PRIMARY,comment_text | PRIMARY,comment_t
```

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%

```
EXPLAIN SELECT comment_id, user_id, comment_text FROM product_comment WHERE comment_text LIKE "%
```

运行结果：

```
+----+-------------+-----------------+------------+------+---------------+------+---------+------+--------+----------+
| id | select_type | table           | partitions | type | possible_keys | key  | key_len | ref  | rows   | filtered | Extra    |
+----+-------------+-----------------+------------+------+---------------+------+---------+------+--------+----------+
|  1 | SIMPLE      | product_comment | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 996663 |    11.11 |
```

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('')。

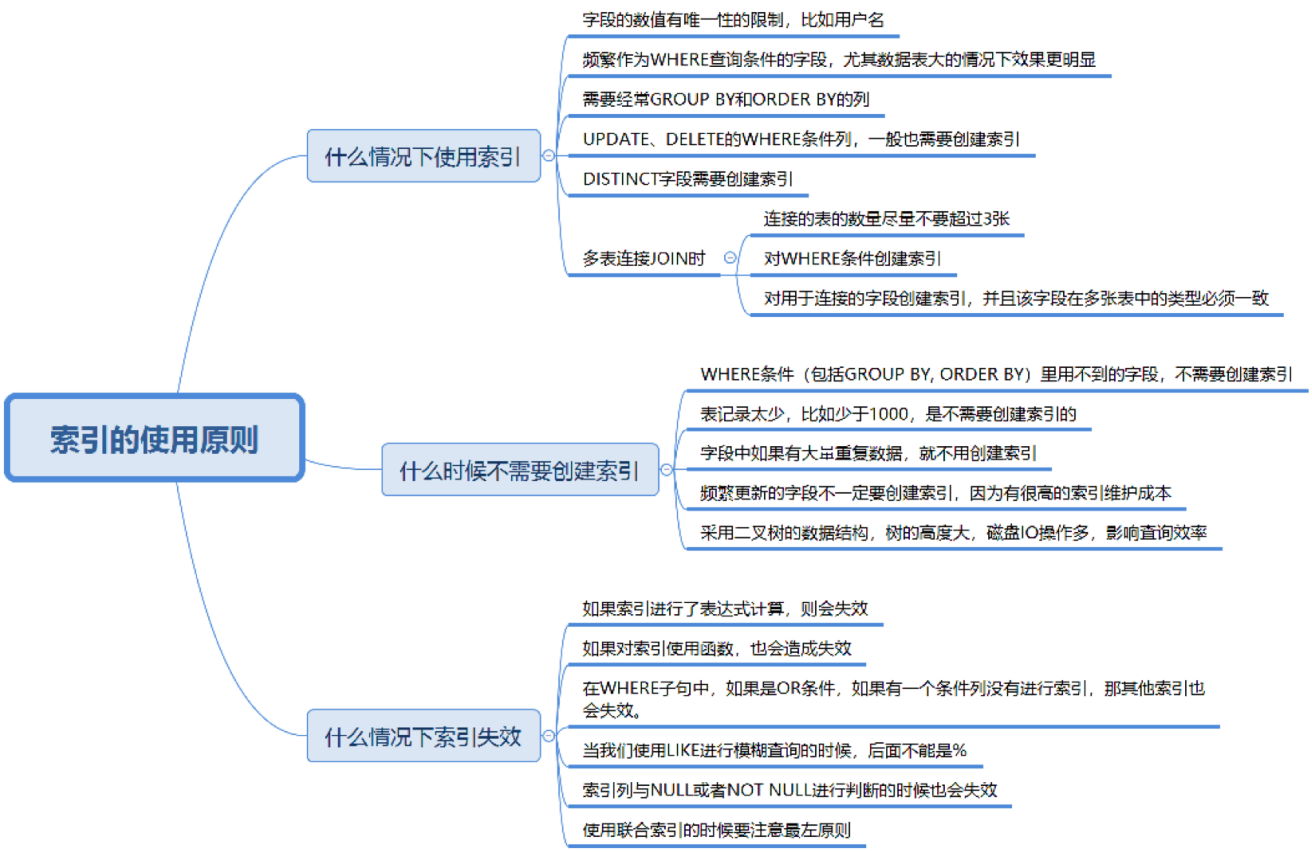
6.我们在使用联合索引的时候要注意最左原则

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：

```
SELECT comment_id, comment_text, comment_time FROM product_comment WHERE DATE(comment_time) >
```

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。

精选留言

我行我素 7

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

2019-08-09

梦想天空 3

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

2019-08-10

haer 2

索引失效，因为使用了date函数

2019-08-09

黑山老妖 1

老师 SELECT user_id, count(*) as num FROM product_comment group by user_id order by c

omment_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 ： ）

2019-08-10

wusiration 1

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')

2019-08-09

niemo 1

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？

2019-08-09

佚花 0

关于like.
%在左边，即使有索引，也会失效.
只有当%在右边时，才会生效

2019-08-21

melon 0

索引列 存null 会导致索引失效 但是 为什么有的时候不会呢？mysql 版本5.7.26
1.show index from heros;
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+-------+------------+----------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| heros | 0 | PRIMARY | 1 | name | A | 69 | NULL | NULL | | BTREE | | |
| heros | 1 | idx_role_assist | 1 | role_assist | A | 6 | NULL | NULL | YES | BTREE | | |

2.explain select * from heros where role_assist is null;
+----+-------------+-------+------------+------+----------------+----------------+---------+-------+------+----------+----------------------+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+----+-------------+-------+------------+------+----------------+----------------+---------+-------+------+----------+----------------------+
| 1 | SIMPLE | heros | NULL | ref | idx_role_assist | idx_role_assist | 768 | const | 40 | 100.00 | Using index condition |

3. explain select * from heros where role_assist is not null;
+----+-------------+-------+------------+------+----------------+------+---------+------+------+----------+-------

------+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------
------+
| 1 | SIMPLE | heros | NULL | ALL | idx_role_assist | NULL | NULL | NULL | 69 | 42.03 | Using where |
麻烦老师 解答一下

2019-08-17

cricket1981 0

建立函数索引

2019-08-16

笃定 0

上述查询中第二次重复查询也会用到数据库的缓存吧，也有索引的作用

2019-08-14

melon 0

老师 为什么group by 后的列要加索引呢？

2019-08-13

刘浩 0

老师好，SELECT user_id, count(*) as num FROM product_comment group by user_id order by comment_time desc limit 100
，单独建索引，查询时间持续100S以上，可以大概解释一下原因吗？

2019-08-13

悟空 0

老师，今天文章中的“product_comment”表结构和数据，是从哪里导入的呢？

个人感觉，本课程用到的所有表都可以放到一个统一的地方，比如之前的 GitHub上面，方便我们统一下载。

2019-08-13

作者回复

因为文件大于100M，就先放到百度网盘：https://pan.baidu.com/s/1LBEAm50DDP9AjErLtGplLg
提取码：32ep
多谢建议，我把它也放到GitHub上，大于100M的给出百度网盘的链接

2019-08-14

Geek_Wison 0

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？

2019-08-09

Ronnyz 0

作业：
对comment_time使用了函数，索引失效
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');

2888 rows in set (1.60 sec)

2019-08-09

ABC 0

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')

手机回复，没有实际运行，如有错误请老师指正，谢谢

2019-08-09

ttttt 0

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：
ERROR 1055 (42000): Expression #1 of ORDER BY clause is not in GROUP BY clause and contains nonaggregated column 'wucai.product_comment.comment_time' which is not functionally dependent on columns in GROUP BY clause; this is incompatible with sql_mode=only_full_group_by

2019-08-09

许童童 0

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')

2019-08-09

lmtoo

0

应该把字符串转为date类型

2019-08-09

LiuChen

0

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？

2019-08-09