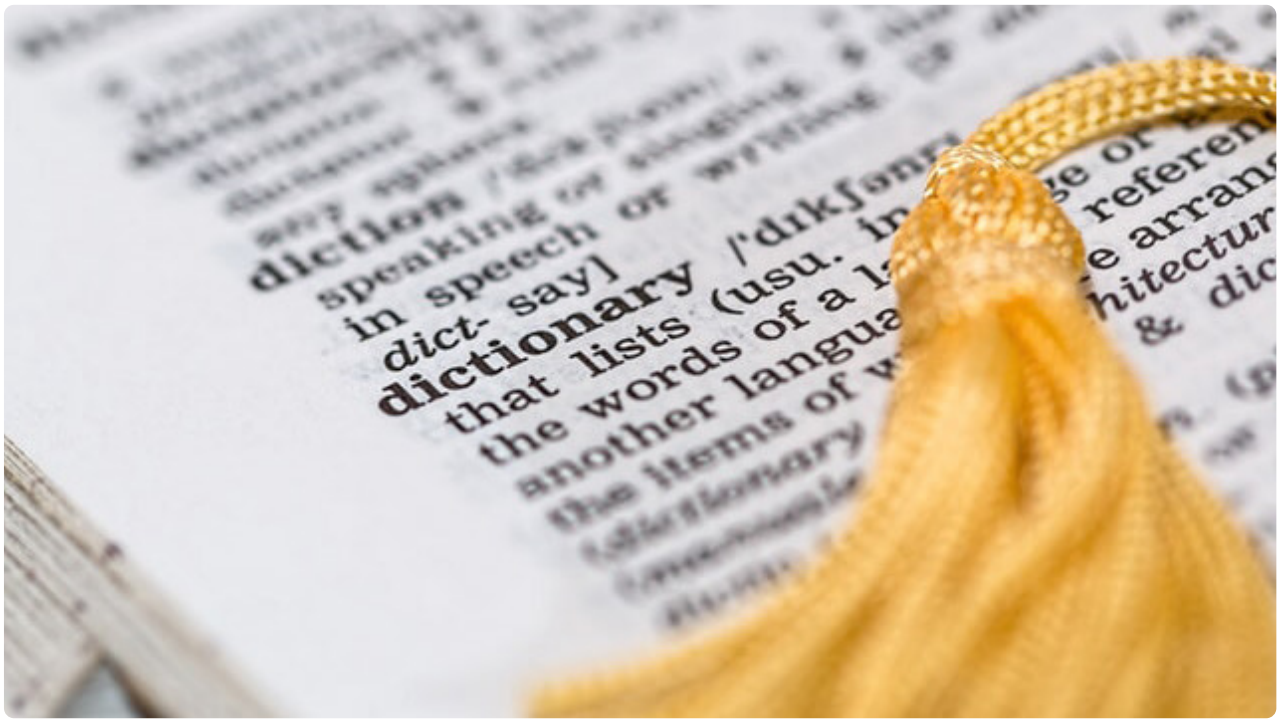
04 条件字段有索引，为什么查询也这么慢?

更新时间：2019-07-30 11:06:17



> 加紧学习，抓住中心，宁精勿杂，宁专勿多。
>
> —— 周恩来

如果我们想在某一本书中找到特定的主题，一般最快的方法是先看索引，找到对应的主题在哪个页码。

而对于 MySQL 而言，如果需要查找某一行的值，可以先通过索引找到对应的值，然后根据索引匹配的记录找到需要查询的数据行。然而，有时会发现，即使查询条件有索引，也会查询很慢，本节将分享这类情况。

MySQL 索引为什么能提高查询速度？将在第二章具体讲解。本节分享的是某些时候有索引却不走索引的情况，当然，大多数情况索引对提升 MySQL 查询速度还是非常明显的。

下面会讲解几种有索引但是查询不走索引导致查询慢的场景。

1 函数操作

小伙伴们在使用 MySQL 查询数据时，可能很多时候会借助一些函数实现查询。有时可能我们关注的重心在是否能查出结果，往往忽略了查询的效率。现在就一起研究对条件索引字段做函数操作，是否能用到索引？

1.1 验证对条件字段做函数操作是否能走索引

首先创建测试表，建表及数据写入语句如下：

```
use muke;                    /* 使用muke这个database */

drop table if exists t1;       /* 如果表t1存在则删除表t1 */

CREATE TABLE `t1` (            /* 创建表t1 */
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `a` varchar(20) DEFAULT NULL,
  `b` int(20) DEFAULT NULL,
  `c` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  KEY `idx_a` (`a`) USING BTREE,
  KEY `idx_b` (`b`) USING BTREE,
  KEY `idx_c` (`c`) USING BTREE
) ENGINE=InnoDB  DEFAULT CHARSET=utf8mb4;

drop procedure if exists insert_t1; /* 如果存在存储过程insert_t1，则删除 */
delimiter ;;
create procedure insert_t1()       /* 创建存储过程insert_t1 */
begin
  declare i int;                   /* 声明变量i */
  set i=1;                         /* 设置i的初始值为1 */
  while(i<=10000)do                /* 对满足i<=10000的值进行while循环 */
    insert into t1(a,b) values(i,i); /* 写入表t1中a、b两个字段，值都为i当前的值 */
    set i=i+1;                     /* 将i加1 */
  end while;
end;;
delimiter ;
call insert_t1();                  /* 运行存储过程insert_t1 */

update t1 set c = '2019-05-22 00:00:00'; /* 更新表t1的c字段，值都为'2019-05-22 00:00:00' */
update t1 set c = '2019-05-21 00:00:00' where id=10000; /* 将id为10000的行的c字段改为与其它行都不一样的数据，以便后面实验使用 */
```

对于上面创建的测试表，比如要查询测试表 t1 单独某一天的所有数据，SQL如下：

```
select * from t1 where date(c) ='2019-05-21';
```

这里就可以使用第 2 节学习的 explain 来分析这条SQL的执行计划，分析结果如下：

```
mysql> explain select * from t1 where date(c) ='2019-05-21';
```

```
mysql> explain select * from t1 where date(c) ='2019-05-21';
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows  | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+-------------+
|  1 | SIMPLE      | t1    | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 10302 |   100.00 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+-------------+
1 row in set, 1 warning (0.01 sec)
```

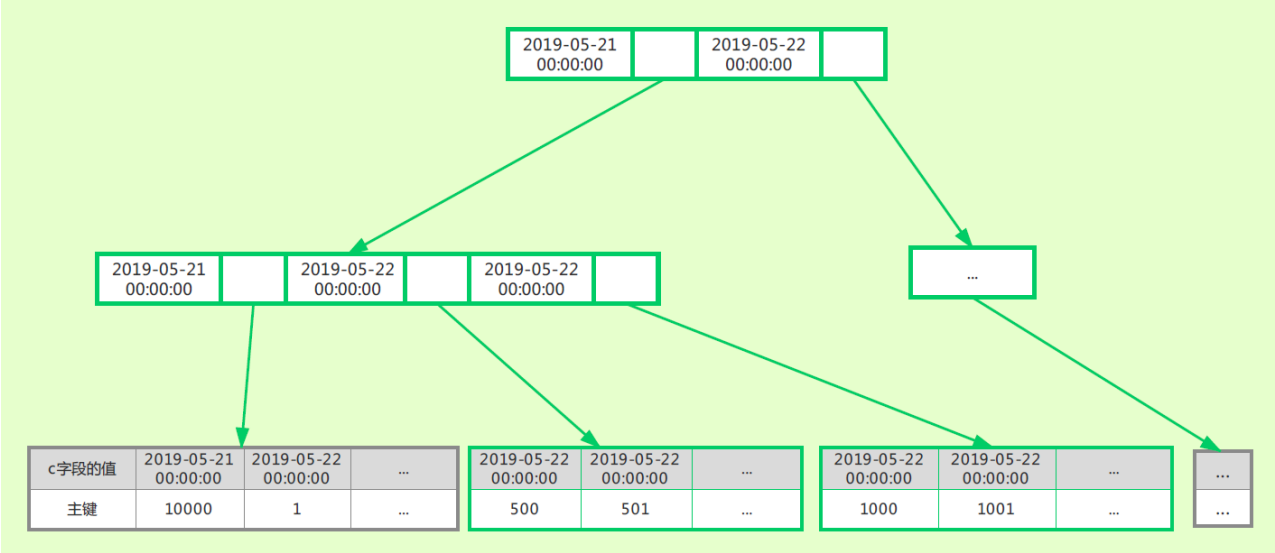
查看图中的执行计划，type 为 ALL，key 字段结果为 NULL，因此知道该 SQL 是没走索引的全表扫描。（执行计划各字段的解释如果忘记了，可以查阅第 2 篇文章《快速学会分析 SQL 执行效率（上）》中的 2 小节）

原因：对条件字段做函数操作走不了索引。

1.2 对条件字段做函数操作不走索引的原因

为什么对条件字段做函数操作走不了索引，我们下面来讨论一下：

该例中 c 字段普通索引的 B+ 索引树如下：



根据上面结构可以看到，索引树中存储的是列的实际值和主键值。如果拿 '2019-05-21' 去匹配，将无法定位到索引树中的值。因此放弃走索引，而选择全表扫描。

1.3 函数操作的 SQL 优化

因此如果需要优化的话，改成 c 字段实际值相匹配的形式。因为 SQL 的目的是查询 2019-05-21 当天所有的记录，因此可以改成范围查询，如下：

```
select * from t1 where c>='2019-05-21 00:00:00' and c<='2019-05-21 23:59:59';
```

再用 explain 分析下执行计划：

```
mysql> explain select * from t1 where c>='2019-05-21 00:00:00' and c<='2019-05-21 23:59:59';
```

```
mysql> explain select * from t1 where c>='2019-05-21 00:00:00' and c<='2019-05-21 23:59:59';
+----+-------------+-------+------------+-------+---------------+-------+---------+------+------+----------+-----------------------+
| id | select_type | table | partitions | type  | possible_keys | key   | key_len | ref  | rows | filtered | Extra                 |
+----+-------------+-------+------------+-------+---------------+-------+---------+------+------+----------+-----------------------+
|  1 | SIMPLE      | t1    | NULL       | range | idx_c         | idx_c | 5       | NULL |    1 |   100.00 | Using index condition |
+----+-------------+-------+------------+-------+---------------+-------+---------+------+------+----------+-----------------------+
1 row in set, 1 warning (0.00 sec)
```

根据上面的结果，可确定，走了 c 字段的索引（对应关注字段 key），扫描行数 1 行（对应关注字段 rows）。

> 经验分享：
>
> 类似求某一天或者某一个月数据的需求，建议写成类似上例的范围查询，可让查询能走索引。避免对条件索引字段做函数处理。
>
> 我在工作中就曾经遇到过这类慢查询，如下：
>
> ```
> SELECT tml_num_id, status_num_id FROM sd_bl_so_tml_hdr WHERE
> tenant_num_id = 6 AND data_sign = 0 AND sub_unit_num_id = 100004 AND
> channel_num_id = 91 AND date_format(order_date, '%Y%m%d') =
> date_format('2019-06-02', '%Y%m%d') AND status_num_id < 3 LIMIT 100;
> ```
>
> 如果明白了上面的优化技巧，可以尝试着改写优化这条 SQL。

2 隐式转换

2.1 认识隐式转换

> 什么时隐式转换？
>
> 当操作符与不同类型的操作对象一起使用时，就会发生类型转换以使操作兼容。某些转换是隐式的。
>
> 关于隐式转换详情请参考MySQL官方手册

隐式转换估计是很多 MySQL 使用者踩过的坑，比如联系方式字段。由于有时电话号码带加、减等特殊字符，有时需要以 0 开头，因此一般设计表时会使用 varchar 类型存储，并且会经常做为条件来查询数据，所以会添加索引。

而有时遇到需要按照手机号码条件（比如　11111111111）去查询数据时，因为查询者看到条件是一串数字，而忽视表中对应手机号字段是 varchar 类型，因此写出了如下不合理的SQL：

```
select user_name,tele_phone from user_info where tele_phone =11111111111; /* SQL 1 */
```

实际情况这条 SQL 查询效率是很低的。首先根据你的经验，思考下这条 SQL 怎么优化？

如果暂时不确定方法，请看下面的实验。

2.2 验证隐式转换是否能走索引

我们一起来通过实验验证一下隐式转换是否能走索引。

实验过程分为：先创建测试表并写入数据；测试隐式转换的查询并查看执行计划；测试正常查询，再查看执行计划。

比如我们要查询 a 字段等于 1000 的值，SQL如下：

```
mysql> select * from t1 where a=1000;
+------+------+------+---------------------+
| id   | a    | b    | c                   |
+------+------+------+---------------------+
| 1000 | 1000 | 1000 | 2019-05-22 00:00:00 |
+------+------+------+---------------------+
1 row in set (0.00 sec)
```

而这条 SQL 是否能使用索引呢？我们一起看下 explain 结果：

```
mysql> explain select * from t1 where a=1000;
```

```
mysql>  explain select * from t1 where a=1000;
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows  | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+-------------+
|  1 | SIMPLE      | t1    | NULL       | ALL  | idx_a         | NULL | NULL    | NULL | 10051 |    10.00 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+-------------+
1 row in set, 3 warnings (0.00 sec)
```

通过 type 这列可以看到是最差的情况 ALL（全表扫描，如果对 explain 结果中 type 各个值没印象的，可以查看第 2 节中<表 3-type 各项值解释>）， 通过 key 这列可以看到没走 a 字段的索引，通过 rows 这列可以看到进行了全表扫描。

2.3 不走索引的原因

a 字段类型是 varchar(20)，而语句中 a 字段条件值没加单引号，导致 MySQL 内部会先把a转换成int型，再去做判断，相当于实际执行的 SQL 语句如下：

```
mysql> select * from t1 where cast(a as signed int) =1000;
```

因此又回到上面说的：**对索引字段做函数操作时，优化器会放弃使用索引。**

2.4 隐式转换的 SQL 优化

索引字符串列条件添加单引号，查看执行计划：

```
mysql> explain select * from t1 where a='1000';
```

```
mysql> explain select * from t1 where a='1000';
+----+-------------+-------+------------+------+---------------+-------+---------+-------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key   | key_len | ref   | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+-------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | t1    | NULL       | ref  | idx_a         | idx_a | 83      | const |    1 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+-------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

通过 type 这列，可以看到是 ref（基于普通索引的等值查询，比 ALL 性能好很多，可复习第 2 节<表3-type 各项值解释>），通过key这列，可以看到已经走了 a 字段的索引，通过rows这列可以看到通过索引查询后就扫描了一行。

因此在联系方式这个例子中的 sql 1 可以这样优化：

```
select user_name,tele_phone from user_info where tele_phone ='11111111111';
```

> **经验分享：**
>
> 隐式转换导致查询慢的情况在工作中遇到过几次，有时字段名对开发写SQL产生了影响，比如曾经遇到过字段名是user_num，而实际字段类型是char，但是开发在写SQL时误认为是int型，导致漏写单引号而发生隐式转换。**所以建议在写SQL时，先看字段类型，然后根据字段类型写SQL。**

3 模糊查询

3.1 分析模糊查询

很多时候我们想根据某个字段的某几个关键字查询数据，比如会有如下 SQL：

```
mysql> select * from t1 where a like '%1111%';
+------+------+------+---------------------+
| id   | a    | b    | c                   |
+------+------+------+---------------------+
| 1111 | 1111 | 1111 | 2019-05-22 00:00:00 |
+------+------+------+---------------------+
1 row in set (0.00 sec)
```

实际这种情况无法走索引，看下执行计划：

```
mysql> explain select * from t1 where a like '%1111%';
```

```
mysql> explain select * from t1 where a like '%1111%';
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows  | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+-------------+
|  1 | SIMPLE      | t1    | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 10051 |    11.11 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

重点留意type、key、rows、Extra，发现是全表扫描。

> Tips：通配符在前面为什么不走索引，将在第二章索引中详细描述。

3.2 模糊查询优化建议

修改业务，让模糊查询必须包含条件字段前面的值，然后落到数据库的查询为：

```
mysql> select * from t1 where a like '1111%';
+------+------+------+---------------------+
| id   | a    | b    | c                   |
+------+------+------+---------------------+
| 1111 | 1111 | 1111 | 2019-05-22 00:00:00 |
+------+------+------+---------------------+
1 row in set (0.00 sec)
```

> Tips：这个优化方式必须结合业务，如果只是这样改SQL，**可能会导致查询的结果不正确**。

这种写法是可以用到索引的，explain分析如下：

```
mysql> explain select * from t1 where a like '1111%';
```

```
mysql> explain select * from t1 where a like '1111%';
+----+-------------+-------+------------+-------+---------------+-------+---------+------+------+----------+-----------------------+
| id | select_type | table | partitions | type  | possible_keys | key   | key_len | ref  | rows | filtered | Extra                 |
+----+-------------+-------+------------+-------+---------------+-------+---------+------+------+----------+-----------------------+
|  1 | SIMPLE      | t1    | NULL       | range | idx_a         | idx_a | 83      | NULL |    1 |   100.00 | Using index condition |
+----+-------------+-------+------------+-------+---------------+-------+---------+------+------+----------+-----------------------+
1 row in set, 1 warning (0.00 sec)
```

> **经验分享：**
>
> 如果条件只知道中间的值，需要模糊查询去查，那就建议使用ElasticSearch或其它搜索服务器。

4 范围查询

也许你会在工作中因为要查询某个范围的数据而使用范围查询，但不知道有没有遇到过这种场景？明明范围查询的条件字段有索引，但是却全表扫描了。

4.1 构造不能使用索引的范围查询

我们拿测试表举例，比如要取出b字段1到2000范围数据，SQL 如下：

```
mysql> select * from t1 where b>=1 and b <=2000;
```

首先看下这条 SQL 的执行计划：

```
mysql> explain select * from t1 where b>=1 and b <=2000;
```

```
mysql> explain select * from t1 where b>=1 and b <=2000;
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows  | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+-------------+
|  1 | SIMPLE      | t1    | NULL       | ALL  | idx_b         | NULL | NULL    | NULL | 10302 |    19.41 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

> 发现并不能走b字段的索引。
>
> 原因：优化器会根据检索比例、表大小、I/O块大小等进行评估是否使用索引。比如单次查询的数据量过大，优化器将不走索引。

4.2 优化范围查询

降低单次查询范围，分多次查询：

```
mysql> select * from t1 where b>=1 and b <=1000;
mysql> select * from t1 where b>=1001 and b <=2000;
```

查看执行计划（就只看第一条的，第二条同理）：

```
mysql> explain select * from t1 where b>=1 and b <=1000;
```

```
mysql> explain select * from t1 where b>=1 and b <=1000;
+----+-------------+-------+------------+-------+---------------+-------+---------+------+------+----------+-----------------------+
| id | select_type | table | partitions | type  | possible_keys | key   | key_len | ref  | rows | filtered | Extra                 |
+----+-------------+-------+------------+-------+---------------+-------+---------+------+------+----------+-----------------------+
|  1 | SIMPLE      | t1    | NULL       | range | idx_b         | idx_b | 5       | NULL | 1000 |   100.00 | Using index condition |
+----+-------------+-------+------------+-------+---------------+-------+---------+------+------+----------+-----------------------+
1 row in set, 1 warning (0.00 sec)
```

因此，降低查询范围后，能正常使用索引。

> 经验分享：
>
> 实际这种范围查询而导致使用不了索引的场景经常出现，比如按照时间段抽取全量数据，每条SQL抽取一个月的；或者某张业务表历史数据的删除。遇到此类操作时，**应该在执行之前对SQL做explain分析，确定能走索引，再进行操作**，否则不但可能导致操作缓慢，在做更新或者删除时，甚至会导致表所有记录锁住，十分危险。

5 计算操作

5.1 查询条件进行计算操作的 SQL 执行效率

有时我们与有对条件字段做计算操作的需求，在使用 SQL 查询时，就应该小心了。先看下例：

```
mysql> explain select * from t1 where b-1 =1000;
```

```
mysql> explain select * from t1 where b-1 =1000;
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows  | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+-------------+
|  1 | SIMPLE      | t1    | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 10302 |   100.00 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

> 原因：对索引字段做运算将使用不了索引。

5.2 计算操作的 SQL 优化

将计算操作放在等号后面：

```
mysql> explain select * from t1 where b =1000 + 1;
```

```
mysql> explain select * from t1 where b =1000 + 1;
+----+-------------+-------+------------+------+---------------+-------+---------+-------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key   | key_len | ref   | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+-------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | t1    | NULL       | ref  | idx_b         | idx_b | 5       | const |    1 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+-------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

发现将计算操作放在等号后，能正常使用索引。

> 经验分享：
>
> **一般需要对条件字段做计算时，建议通过程序代码实现，而不是通过MySQL实现。如果在MySQL中计算的情况避免不了，那必须把计算放在等号后面。**

6 总结

本节讲解几种条件字段有索引，但是使用不了索引的场景。因此在写 SQL 时应该注意这些点：

- 应该避免隐式转换
- like查询不能以%开头
- 范围查询时，包含的数据比例不能太大
- 不建议对条件字段做运算及函数操作

本节涉及到的一些SQL优化如下图：

7 问题

你在工作中遇到过哪些隐式转换的情况？

8 参考资料

MySQL 5.7参考手册

https://dev.mysql.com/doc/refman/5.7/en/type-conversion.html

}

03 快速学会分析SQL执行效率（下）

05 如何优化数据导入？