# 18\_为什么这些SQL语句逻辑相同，性能却差异巨大

**案例一 条件字段函数作用**

维护一个交易系统，其中交易记录表tradelog包含交易流水号 (tradeid)、交易员id(operator)、交易时间（t\_modified）等字段，建表语句如下

mysql> CREATE TABLE `tradelog` (

`id` int(11) NOT NULL,

`tradeid` varchar(32) DEFAULT NULL,

`operator` int(11) DEFAULT NULL,

`t\_modified` datetime DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `tradeid` (`tradeid`),

KEY `t\_modified` (`t\_modified`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

假设，现在已经有记录从2016年初到2018年底的所有数据，运营部门有个需求，要统计发生在所有年份的交易记录总数，SQL不复杂，SQL可能会这么写

mysql> select count(\*) from tradelog where month(t\_modified)=7;

由于t\_modified字段上有索引，于是很放心的生产上使用这条语句，却发现执行特别久

**如果对字段使用了函数计算，就用不上索引了，这是MySQL规定**

为什么where t\_modeified=’2018-7-1’可以用上索引，而改成where moth(t\_modified)=7 的时候就不行了。

下面是这个t\_Modefied索引的示意图，方框上的数字就是month()函数对应的值

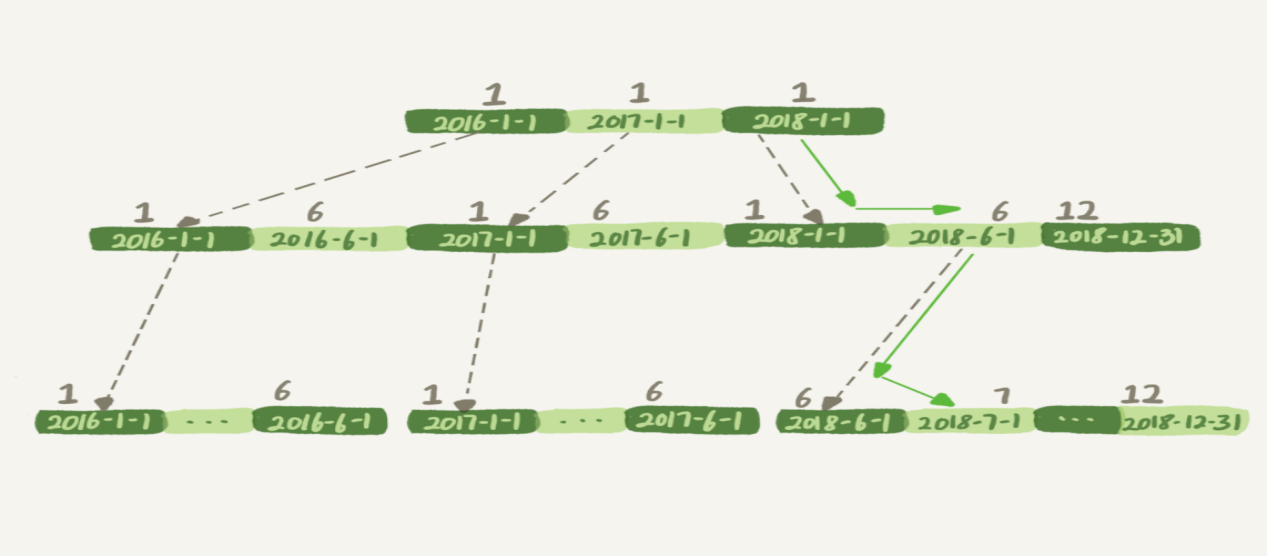


图 1 t\_modified索引示意图

如果SQL语句条件是where t\_modified=’2018-7-1’的话，引擎就会按照上面绿色箭头的路线，快速定位到t\_modified=’2018-7-1’需要的结果

**实际上，B+数提供的快速定位能力，来源于同一层兄弟节点的有序性**

但是，如果计算month()函数的话，会看到传入7的时候，在树的第一层就不知道该怎么办了

也就是说，**对索引字段做函数操作，可能会破坏索引值的有序性，因此优化器就决定放弃走树搜索功能。**

**\*\*注意，优化器并不是要放弃使用这个索引。**

在这个例子里，放弃数搜索功能，优化器可以选择遍历主键索引，也可以遍历索引t\_modified,优化器对比索引大小后发现，索引t\_modified更小，遍历这个索引比遍历主键索引来得更快，因此最终还是会选择t\_modified.

使用explain ,查看执行结果

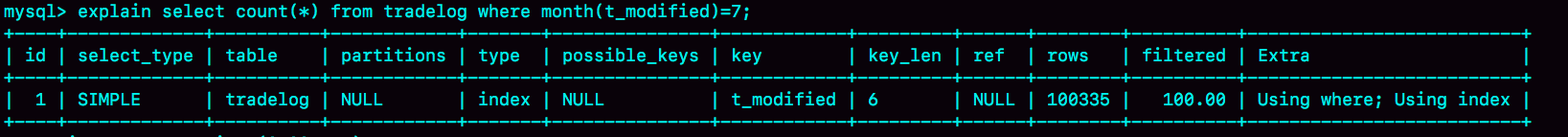


图 2 explain 结果

Key=t\_modified 表示使用了t\_modified这个索引；测试表数据插入10万行数据，rows=100335，说明这条语句扫描了整个索引的所有值，Extraz字段Using index ,表示使用了覆盖索引

由于t\_modified 字段加了month()函数操作，导致了全表扫描，为了能够用上索引的快速定位能力，我们把SQL语句改为基于字段本身的范文查找，按照下面这个写法，优化器能搞按照预期，用上t\_modified快速定位能力

mysql> select count(\*) from tradelog where

-> (t\_modified >= '2016-7-1' and t\_modified<'2016-8-1') or

-> (t\_modified >= '2017-7-1' and t\_modified<'2017-8-1') or

-> (t\_modified >= '2018-7-1' and t\_modified<'2018-8-1');

比如 select \* from tradelog where id + 1= 10000 这个语句，这个加1操作并不会改变有序性，但MySQL优化器还是不能使用id 索引快速定位到9999这一行，所以需要改写SQL

Select \* from tradelog where id = 9999;

**案例二 隐式类型转换**

下一条SQL

mysql> select \* from tradelog where tradeid=110717;

交易编号tradeid这个字段上，本来有索引，但是explain的结果却显示，这条语句需要走全表扫描，tradeid的字段类型varchar(32),而输入的参数是整型，所以需要类型转换

两个问题

1 数据类型转换的规则时什么？

2 为什么有类型转换，就需要走全表扫描？

一个简单的方法 看select “10” > 9 的结果

1 如果规则是“将字符串转换成数字”，那么就是数字作比较，结果应该是1

2 如果规则是“将数字转换成字符串”，那么就是字符串比较，结果应该是0；

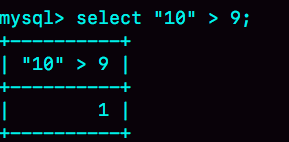


图3 MySQL 字符串和数字转换的示意图

图中可知，select “10” > 9 返回的是1，**所以在MySQL中，字符串和数字比较的话，是将字符串转换成数字**

**mysql> select \* from tradelog where tradeid=110717;**

**对于优化器来说，这个语句相当于**

mysql> select \* from tradelog where CAST(tradid AS signed int) = 110717;

也就是说，对索引字段做函数操作，优化器会放弃走树搜索功能

小问题 id 的类型是int,如果执行下面语句是否会导致全表扫描呢？

select \* from tradelog where id="83126";

**案例三 隐式字符编码转化**

假设系统里还有另外一个trade\_detail，用于交易记录的操作细节，为了便于量化分析和复现，交易日志表tradelog和交易详情表trade\_detail这两个表里插入一些数据

mysql> CREATE TABLE `trade\_detail` (

`id` int(11) NOT NULL,

`tradeid` varchar(32) DEFAULT NULL,

`trade\_step` int(11) DEFAULT NULL, /\* 操作步骤 \*/

`step\_info` varchar(32) DEFAULT NULL, /\* 步骤信息 \*/

PRIMARY KEY (`id`),

KEY `tradeid` (`tradeid`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

insert into tradelog values(1, 'aaaaaaaa', 1000, now());

insert into tradelog values(2, 'aaaaaaab', 1000, now());

insert into tradelog values(3, 'aaaaaaac', 1000, now());

insert into trade\_detail values(1, 'aaaaaaaa', 1, 'add');

insert into trade\_detail values(2, 'aaaaaaaa', 2, 'update');

insert into trade\_detail values(3, 'aaaaaaaa', 3, 'commit');

insert into trade\_detail values(4, 'aaaaaaab', 1, 'add');

insert into trade\_detail values(5, 'aaaaaaab', 2, 'update');

insert into trade\_detail values(6, 'aaaaaaab', 3, 'update again');

insert into trade\_detail values(7, 'aaaaaaab', 4, 'commit');

insert into trade\_detail values(8, 'aaaaaaac', 1, 'add');

insert into trade\_detail values(9, 'aaaaaaac', 2, 'update');

insert into trade\_detail values(10, 'aaaaaaac', 3, 'update again');

insert into trade\_detail values(11, 'aaaaaaac', 4, 'commit');

这时候，如果查询ID=2的交易所有的操作信息，SQL语句可以这么写

mysql> select d.\* from tradelog l, trade\_detail d where d.tradeid=l.tradeid and l.id=2; /\* 语句 Q1\*/

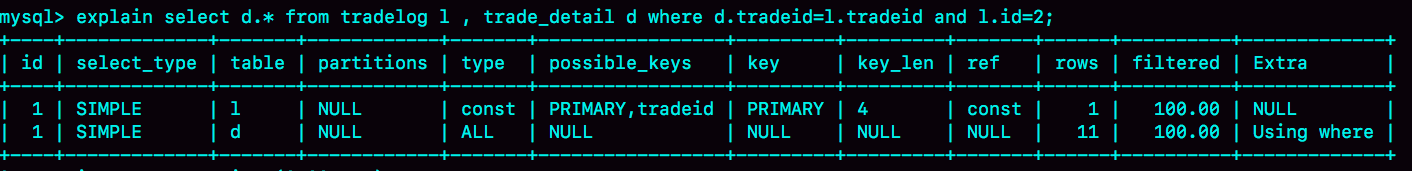


图 4 语句Q1的expalin结果

查看结果

1. 第一行显示优化器会先在交易记录表tradelog上查到id=2的记录，这个步骤用上了主键索引rows=1,表示只扫描了一行
2. 第二行key= NULL，表示没有用上交易详情表trade\_detail上的tradeid索引，进行全表扫描

在这个执行计划里，从tradelog表中取tradeid字段，再去trade\_detail表里查询匹配字段，因此用tradelog作为驱动表，把trade\_detail称为被驱动表，把tradeid称为关联字段。

查看explain 结果的执行流程

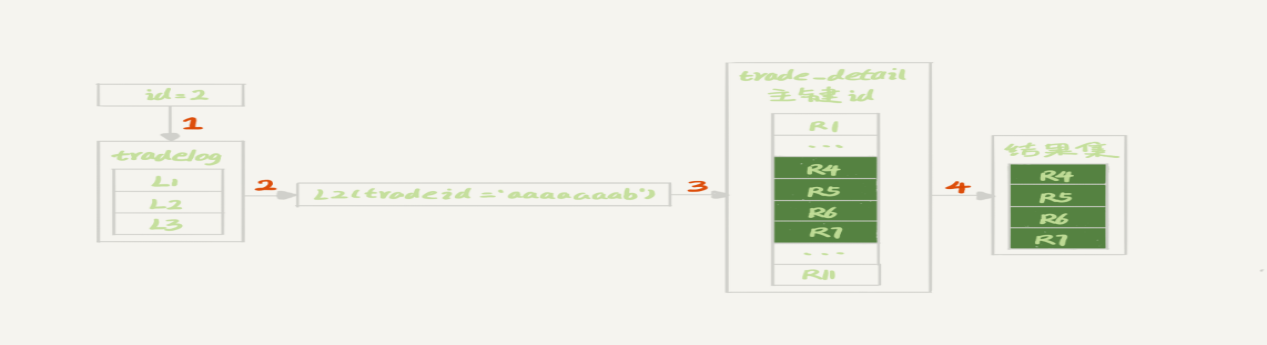


图 5 Q 1的执行过程

·第1步，是根据id在tradelog表里找到L2这一行

·第2步，从L2中取tradeid字段

·第3步，是根据tradeid值到trade\_detail表中查找条件匹配的行，explain的结果里面第二行key=NULL表示的就是，这个过程通过遍历主键索引的方式，一个一个判断tradeid的值是否匹配。

1. 并没有走索引tradeid字段，因为两个表的字符集不同，一个是utf8,一个是utf8mb4,所以表连接查询时用不上关联字段索引，

问题出在第3步，如果单独把这一步改成SQL语句

mysql> select \* from trade\_detail where tradeid=$L2.tradeid.value;

其中 $L2.tradeid.value;的字符集是utf8mb4,

参照前面的例子，字符集utf8mb4是utf8的超集，所以当两个类型的字符串作比较的时候，MySQL内部的操作是，先把utf8字符串转为utf8mb4,在作比较

这个设定很好理解，utf8mb4是utf8的超集，类似地，做自动类型转化的时候，为了避免数据再转换过程中由于截断导致数据错误，也都是“按数据长度增加的方向”进行转换

因此在执行上面这个语句的时候，需要将驱动数据表里的字段一个个的转换成utf8mb4在跟L2做比较

也就是说，上面的语句等于下面的写法

select \* from trade\_detail where CONVERT(traideid USING utf8mb4)=$L2.tradeid.value;

CONVERT()函数，在这里的意思是把输入的字符串转成utf8mb4字符集

再次出发上面说的原则**：对索引字段做函数操作，优化器会放弃走索引树的搜索功能。**

**字符集不同只是条件之一，连接过程中要求在被驱动表的索引字段上加函数操作，**是直接导致对驱动表做全表扫描的原因。

作为验证对比，“查找trade\_detail”表里id=4的操作，对应的操作是谁，再来看下这个语句和他的执行计划

mysql>select l.operator from tradelog l , trade\_detail d where d.tradeid=l.tradeid and d.id=4;

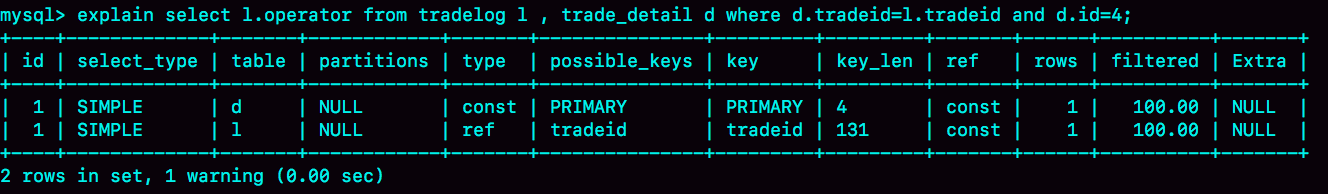


图 6 explain 结果

这个语句里trade\_detail 是驱动表，但是explain结果的第二行显示，这次查询的操作用上了被驱动表tradelog里的索引（tradeid）,扫描行数是1.

这也是两个tradeid字段的join操作，为什么这次能用上被驱动表的tradeid索引呢？

假设驱动表trade\_detail里id=4的行记录是R4,那么连接的时候（图5的第3步，）被驱动表tradelog上执行的就是类似的这样的语句

select operator from tradelog where traideid =$R4.tradeid.value;

此时，$R4.tradeid.value的字符集是utf8按照字符集的转化规则，要转成utf8mb4,所以这个过程就被改写

Select operator from tradelog where tradeid=CONVERT($R4.tradeid.value USING utf8mb4)

这里 的CONVERT 函数是加载输入参数上的饿，这样可以用上被驱动表的tradeid索引，

如果要优化语句

select d.\* from tradelog l, trade\_detail d where d.tradeid=l.tradeid and l.id=2;

的执行过程，有两种做法

比较常用的优化方法是 trade\_detail表上的tradeid字段的字符集改成utf8mb4,这样就没有字符集转换的问题了

alter table trade\_detail modify tradeid varchar(32) CHARACTER SET utf8mb4 default null;

如果能够修改字段的字符集的话，是最好的。但是数据量比较大，或者业务上暂时不能做这个DDL的话，只能采用修改SQL语句的方法

mysql> select d.\* from tradelog l , trade\_detail d where d.tradeid=CONVERT(l.tradeid USING utf8) and l.id=2;

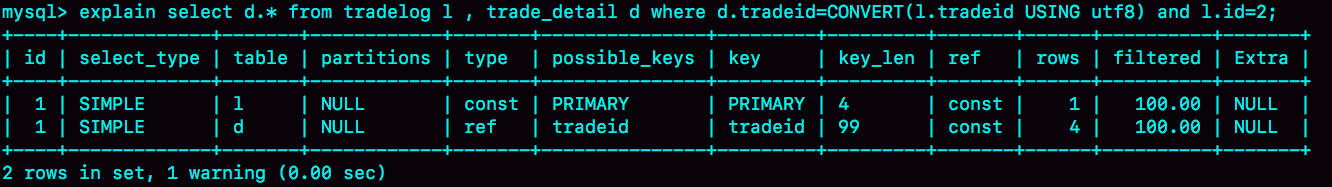


图 7 SQL 语句优化后的explain的结果

把l.tradeid转成utf8,避免了被驱动表上的字符编码转换，可以看到走索引了。