35 | MySQL调优之索引：索引的失效与优化

2019-08-10 刘超

你好，我是刘超。

不知道你是否跟我有过同样的经历，那就是作为一个开发工程师，经常被DBA叫过去“批评”，而最常见的就是申请创建新的索引或发现慢SQL日志了。

记得之前有一次迭代一个业务模块的开发，涉及到了一个新的查询业务，需要根据商品类型、订单状态筛选出需要的订单，并以订单时间进行排序。由于sku的索引已经存在了，我在完成业务开发之后，提交了一个创建status的索引的需求，理由是SQL查询需要使用到这两个索引：

> select * from order where status =1 and sku=10001 order by create_time asc

然而，DBA很快就将这个需求驳回了，并给出了重建一个sku、status以及create_time组合索引的建议，查询顺序也改成了 sku=10001 and status=1。当时我是知道为什么要重建组合索引，但却无法理解为什么要添加create_time这列进行组合。

从执行计划中，我们可以发现使用到了索引，那为什么DBA还要求将create_time这一列加入到组合索引中呢？这个问题我们在第32讲中提到过，相信你也已经知道答案了。通过故事我们可以发现索引知识在平时开发时的重要性，然而它又很容易被我们忽略，所以今天我们就来详细聊一聊索引。

MySQL索引存储结构

索引是优化数据库查询最重要的方式之一，它是在MySQL的存储引擎层中实现的，所以每一种存储引擎对应的索引不一定相同。我们可以通过下面这张表格，看看不同的存储引擎分别支持哪种索引类型：

索引类型	MyISAM引擎	InnoDB引擎	Memory引擎
B+Tree索引	yes	yes	yes
HASH索引	no	no	yes
R-Tree索引	yes	no	no
Full-Text索引	yes	no	no

B+Tree索引和Hash索引是我们比较常用的两个索引数据存储结构，B+Tree索引是通过B+树实现的，是有序排列存储，所以在排序和范围查找方面都比较有优势。如果你对B+Tree索引不够了解，可以通过该链接了解下它的数据结构原理。

Hash索引相对简单些，只有Memory存储引擎支持Hash索引。Hash索引适合key-value键值对查询，无论表数据多大，查询数据的复杂度都是O(1)，且直接通过Hash索引查询的性能比其它索引都要优越。

在创建表时，无论使用InnoDB还是MyISAM存储引擎，默认都会创建一个主键索引，而创建的主键索引默认使用的是B+Tree索引。不过虽然这两个存储引擎都支持B+Tree索引，但它们在具体的数据存储结构方面却有所不同。

InnoDB默认创建的主键索引是聚簇索引（Clustered Index），其它索引都属于辅助索引（Secondary Index），也被称为二级索引或非聚簇索引。接下来我们通过一个简单的例子，说明下这两种索引在存储数据中的具体实现。

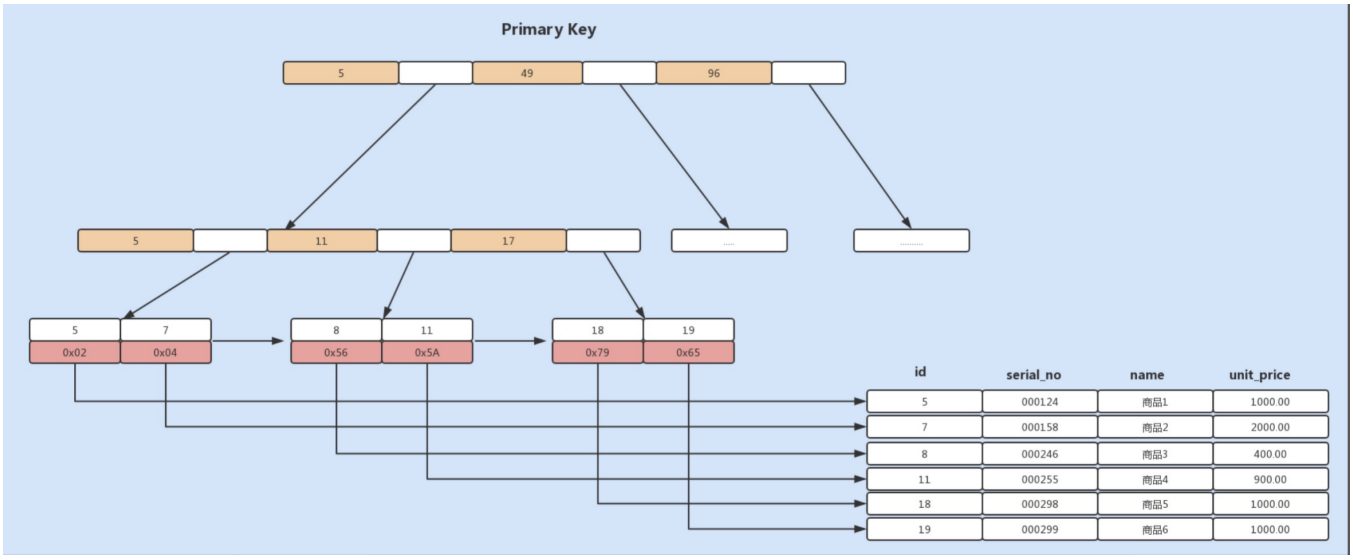
首先创建一张商品表，如下：

```
CREATE TABLE `merchandise` (
  `id` int(11) NOT NULL,
  `serial_no` varchar(20)  DEFAULT NULL,
  `name` varchar(255) DEFAULT NULL,
  `unit_price` decimal(10, 2) DEFAULT NULL,
  PRIMARY KEY (`id`) USING BTREE
) CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT = Dynamic;
```

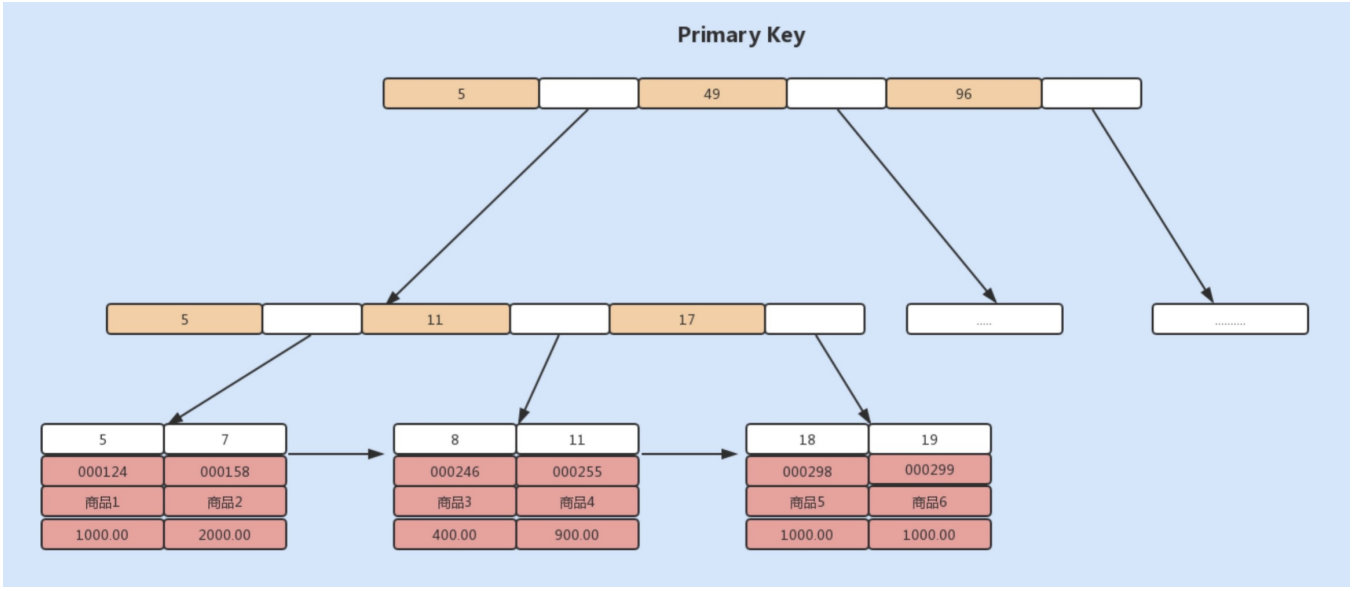
然后新增了以下几行数据，如下：

id	serial_no	name	unit_price
5	000124	商品1	1000.00
7	000158	商品2	2000.00
8	000246	商品3	400.00
11	000255	商品4	900.00
14	000298	商品5	1000.00

如果我们使用的是MyISAM存储引擎，由于MyISAM使用的是辅助索引，索引中每一个叶子节点仅仅记录的是每行数据的物理地址，即行指针，如下图所示：



如果我们使用的是InnoDB存储引擎，由于InnoDB使用的是聚簇索引，聚簇索引中的叶子节点则记录了主键值、事务id、用于事务和MVCC的回流指针以及所有的剩余列，如下图所示：



基于上面的图示，如果我们需要根据商品编码查询商品，我们就需要将商品编码serial_no列作为一个索引列。此时创建的索引是一个辅助索引，与MyISAM存储引擎的主键索引的存储方式是一

致的，但叶子节点存储的就不是行指针了，而是主键值，并以此来作为指向行的指针。这样的好处就是当行发生移动或者数据分裂时，不用再维护索引的变更。

如果我们使用主键索引查询商品，则会按照B+树的索引找到对应的叶子节点，直接获取到行数据：

> select * from merchandise where id=7

如果我们使用商品编码查询商品，即使用辅助索引进行查询，则会先检索辅助索引中的B+树的serial_no，找到对应的叶子节点，获取主键值，然后再通过聚簇索引中的B+树检索到对应的叶子节点，然后获取整行数据。这个过程叫做回表。

在了解了索引的实现原理后，我们再来详细了解下平时建立和使用索引时，都有哪些调优方法呢？

1.覆盖索引优化查询

假设我们只需要查询商品的名称、价格信息，我们有什么方式来避免回表呢？我们可以建立一个组合索引，即商品编码、名称、价格作为一个组合索引。如果索引中存在这些数据，查询将不会再次检索主键索引，从而避免回表。

从辅助索引中查询得到记录，而不需要通过聚簇索引查询获得，MySQL中将其称为覆盖索引。使用覆盖索引的好处很明显，我们不需要查询出包含整行记录的所有信息，因此可以减少大量的I/O操作。

通常在InnoDB中，除了查询部分字段可以使用覆盖索引来优化查询性能之外，统计数量也会用到。例如，在第32讲我们讲 SELECT COUNT(*)时，如果不存在辅助索引，此时会通过查询聚簇索引来统计行数，如果此时正好存在一个辅助索引，则会通过查询辅助索引来统计行数，减少I/O操作。

通过EXPLAIN，我们可以看到 InnoDB 存储引擎使用了idx_order索引列来统计行数，如下图所示：

```
1  select count(*) from  `demo`.`order`
2
```

信息 | Explain 1 | Result 1 | 概况 | 状态

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	order	(Null)	index	(Null)	idx_order	6	(Null)	180819	100	Using index

2.自增字段作主键优化查询

上面我们讲了 InnoDB 创建主键索引默认为聚簇索引，数据被存放在了B+树的叶子节点上。也就是说，同一个叶子节点内的各个数据是按主键顺序存放的，因此，每当有一条新的数据插入时，数据库会根据主键将其插入到对应的叶子节点中。

如果我们使用自增主键，那么每次插入的新数据就会按顺序添加到当前索引节点的位置，不需要移动已有的数据，当页面写满，就会自动开辟一个新页面。因为不需要重新移动数据，因此这种插入数据的方法效率非常高。

如果我们使用非自增主键，由于每次插入主键的索引值都是随机的，因此每次插入新的数据时，就可能会插入到现有数据页中间的某个位置，这将不得不移动其它数据来满足新数据的插入，甚至需要从一个页面复制数据到另外一个页面，我们通常将这种情况称为页分裂。页分裂还有可能会造成大量的内存碎片，导致索引结构不紧凑，从而影响查询效率。

因此，在使用InnoDB存储引擎时，如果没有特别的业务需求，建议使用自增字段作为主键。

3.前缀索引优化

前缀索引顾名思义就是使用某个字段中字符串的前几个字符建立索引，那我们为什么需要使用前缀来建立索引呢？

我们知道，索引文件是存储在磁盘中的，而磁盘中最小分配单元是页，通常一个页的默认大小为16KB，假设我们建立的索引的每个索引值大小为2KB，则在一个页中，我们能记录8个索引值，假设我们有8000行记录，则需要1000个页来存储索引。如果我们使用该索引查询数据，可能需要遍历大量页，这显然会降低查询效率。

减小索引字段大小，可以增加一个页中存储的索引项，有效提高索引的查询速度。在一些大字符串的字段作为索引时，使用前缀索引可以帮助我们减小索引项的大小。

不过，前缀索引是有一定的局限性的，例如order by就无法使用前缀索引，无法把前缀索引用作覆盖索引。

4.防止索引失效

当我们习惯建立索引来实现查询SQL的性能优化后，是不是就万事大吉了呢？当然不是，有时候我们看似使用到了索引，但实际上并没有被优化器选择使用。

对于Hash索引实现的列，如果使用到范围查询，那么该索引将无法被优化器使用到。也就是说Memory引擎实现的Hash索引只有在“=”的查询条件下，索引才会生效。我们将order表设置为Memory存储引擎，分析查询条件为id<10的SQL，可以发现没有使用到索引。

```
1  EXPLAIN SELECT * FROM `order` where id < 10;
2
```

信息 | Result 1 | 概况 | 状态

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	order	(Null)	ALL	PRIMARY	(Null)	(Null)	(Null)	581	33.33	Using whe

如果是以%开头的LIKE查询将无法利用节点查询数据：

```
EXPLAIN SELECT * FROM `order` where order_no like '%1'
```

信息 | Result 1 | 概况 | 状态

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	order	(Null)	ALL	(Null)	(Null)	(Null)	(Null)	581	11.11	Using where

当我们在使用复合索引时，需要使用索引中的最左边的列进行查询，才能使用到复合索引。例如我们在order表中建立一个复合索引idx_user_order_status(order_no, status, user_id)，如果我们使用order_no、order_no+status、order_no+status+user_id以及order_no+user_id组合查询，则能利用到索引；而如果我们用status、status+user_id查询，将无法使用到索引，这也是我们经常听过的最左匹配原则。

```
EXPLAIN SELECT * FROM `order` where order_no='' and user_id=1
```

信息 | Result 1 | 概况 | 状态

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	order	(Null)	ref	idx_order,idx_user_ord	idx_order	5	const	1	10	Using where

```
EXPLAIN SELECT * FROM `order` where status=1 and user_id=1
```

信息 | Result 1 | 概况 | 状态

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	order	(Null)	ALL	(Null)	(Null)	(Null)	(Null)	581	1	Using where

如果查询条件中使用or，且or的前后条件中有一个列没有索引，那么涉及的索引都不会被使用到。

```
EXPLAIN SELECT * FROM `order` where order_no = '1' or create_date = '';
```

信息 | Result 1 | 概况 | 状态

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	order	(Null)	ALL	idx_order	(Null)	(Null)	(Null)	581	19	Using where

所以，你懂了吗？作为一名开发人员，如果没有熟悉MySQL，特别是MySQL索引的基础知识，很多时候都将被DBA批评到怀疑人生。

总结

在大多数情况下，我们习惯使用默认的 InnoDB 作为表存储引擎。在使用InnoDB作为存储引擎时，创建的索引默认为B+树数据结构，如果是主键索引，则属于聚簇索引，非主键索引则属于辅助索引。基于主键查询可以直接获取到行信息，而基于辅助索引作为查询条件，则需要进行回表，然后再通过主键索引获取到数据。

如果只是查询一列或少部分列的信息，我们可以基于覆盖索引来避免回表。覆盖索引只需要读取索引，且由于索引是顺序存储，对于范围或排序查询来说，可以极大地极少磁盘I/O操作。

除了了解索引的具体实现和一些特性，我们还需要注意索引失效的情况发生。如果觉得这些规则

太多，难以记住，我们就要养成经常检查SQL执行计划的习惯。

思考题

假设我们有一个订单表order_detail，其中有主键id、主订单order_id、商品sku等字段，其中该表有主键索引、主订单id索引。

现在有一个查询订单详情的SQL如下，查询订单号范围在5000~10000，请问该查询选择的索引是什么？有什么方式可以强制使用我们期望的索引呢？

```
select * from order_detail where order_id between 5000 and 10000;
```

期待在留言区看到你的答案。也欢迎你点击“请朋友读”，把今天的内容分享给身边的朋友，邀请他一起讨论。

精选留言

QQ怪

14

回答老师问题：

按照老师的操作了一遍，实验小部分区间是会走order_id索引的，但是查询范围继续增大，反而不走索引而是全表扫描，大概我估摸着的是小于全表5分之一区间能够走索引，超过5分之一会全表扫描，可以使用force index（索引名）强制使用该索引，这就是有些sql表开始跑的挺快

的，后面越来越慢的原因吧。。但不清楚mysql优化器为啥要这样选择，希望老师解惑？

2019-08-10

作者回复

因为order_id索引不能覆盖我们要查询的信息，所以在对order_id查询之后还需要一次回表来查找到整行数据，虽然order_id索引是顺序存放的，但是相对于主键id存放的数据顺序是不一致的，所以存在每次回表都是随机获取整行数据，如果在获取大量数据时，通过这种方式获取数据性能肯定是不理想的。

所以mysql一般判断在查询超过整个表20%的数据时，就会考虑使用聚族索引来查找数据，这种方式顺序读取数据的可能性要大于使用辅助索引的随机读。

在查询少量数据的情况下，使用辅助索引性能更加，而查询大量数据时，就未必了。

如果我们发现在查询一定量数据使用辅助索引要比主键索引快，而数据库又没有按照我们期望的去使用辅助索引，则我们可以通过子查询或force index来强制使用辅助索引。

2019-08-12

CCC 4

对索引进行函数操作或者表达式计算也会导致索引的失效

2019-08-10

作者回复

对的，点赞补充

2019-08-12

Charles 3

想问下老师为什么回表查询的速度会慢于直接用主键查询，因为回表也是使用主键ID去查询的，就算查询的数据量大，用不用子查询都是使用主键ID去回表或是查询，速度应该一样吧

2019-08-14

作者回复

回表就相当于两次索引树扫描操作了，而主键查询只有一次。

2019-09-15

Geek__ad4af7fe01f4 2

请问老师，既然使用辅助索引效率低，mysql默认超出20%又使用主键索引优化，而优化的效果又变低，为何还要强制使用辅助索引？

这里强制使用辅助索引的优化 和下面您的描述不是冲突吗？

因为order_id索引不能覆盖我们要查询的信息，所以在对order_id查询之后还需要一次回表来查找到整行数据，虽然order_id索引是顺序存放的，但是相对于主键id存放的数据顺序是不一致的，所以存在每次回表都是随机获取整行数据，如果在获取大量数据时，通过这种方式获取数据性能肯定是不理想

2019-08-27

作者回复

这个跟具体场景有关系，在数据量非常大的情况下，可能使用辅助索引会效率更高些。

2019-08-28

张三丰

1

”如果不存在辅助索引，此时会通过查询聚簇索引来统计行数，如果此时正好存在一个辅助索引，则会通过查询辅助索引来统计行数，减少 I/O 操作。“

这有什么区别吗？都是通过索引统计行数

2019-10-04

作者回复

区别在于聚簇索引存储了其他数据，而辅助索引只保存了索引列和主键，所以通过查询辅助索引统计行检索的数据量会更少，I/O操作会更少

2019-10-06

张学磊

1

由于是select *操作，所以每条记录都需进行回表，当server层分析器发现between的范围太大时，使用辅助索引存在大量回表操作，所以觉得得不偿失，故而直接使用主键索引。如果想使用我们期望的索引，需要给server层分析器一个hint，force index(idx_order_id)

2019-08-10

作者回复

分析到位，答案正确。

2019-08-12

风轻扬

0

老师，我试了一下最左匹配原则。按照您的例子试了一下，发现有的时候，and两端的表达式交换顺序，依然可以使用到复合索引。查了查，是因为mysql的优化器会自己优化。这是怎么回事呢？网上的解释没有看懂。。。。。。

2019-09-23

man1s

0

走主键索引，优化器认为5000数据+回表5000次性能消耗要大于全表扫描

force index

2019-09-16

作者回复

[]

2019-09-17

Demon.Lee

0

Key Point:

如果觉得这些规则太多，难以记住，我们就要养成经常检查 SQL 执行计划的习惯。

2019-09-16

godtrue

0

老师，请问存储引擎具体判断使用什么索引的原则是啥?大体的原则肯定是怎么快怎么走?不过也存在一定的误判，请问老师清楚误判的原因和具体都有那些场景嘛？

2019-09-14

作者回复

mysql查询优化是基于检索成本考虑，而不是基于时间成本考虑，假设我们读取的是随机行的数据，在磁盘中存储是无序的，有可能在扫描少数行的情况下，所需时间更长，这种情况下会出现误选择索引。

通常在一些in操作时，在数据量比较小的情况下，会使用我们建立的索引，当数据量超过一定量时，会改走主键索引。我们一般是通过慢日志来排查这些问题，一旦发现不是走的我们想要的索引，可以使用force index来强制走期望的索引。

2019-09-15

DY

0

老师，你好。 select * from order_detail where id in (select id from order_detail where order_id between 5000 and 10000)； 这种优化方式我试了试， 没起到什么优化作用。问了下DBA，说都回表了， 先查询主键ID也回表， 感觉和自己理解的不一样，但是又没法反驳。看了下执行计划这条sql也确实回表了

2019-09-10

作者回复

我们这里讨论的是索引失效

2019-09-10

疯狂咸鱼

0

老师，你这个表里的order id和id不是一起递增的么？如果orderid也是递增的，那情况又是怎呢

2019-09-07

作者回复

如果与id索引的排序是一致的，会走索引，可以动手实践一下

2019-09-10

我行我素

0

老师，想请问下InnoDB引擎下使用HASH索引也可以啊，但是文中的图InnoDB索引Hash是no

2019-08-13

作者回复

官网给出的是不支持自创建hash数据结构的索引，但是它是自适应的，也就是我们不能人为的干预使用hash索引。具体的可以参考官网：https://dev.mysql.com/doc/refman/8.0/en/create-index.html

2019-08-13

胡峣

0

好几个都同学都提到 select * from order_detail where id in (select id from order_detail where or

der_id between 5000 and 10000)，至少mysql 57里这样写

2019-08-13

密码123456 0

我这一直用的都是oracle，看到mysql就想，快快跳过。后来发现，和数据库关系不大，很多都是通用的。

2019-08-12

作者回复

多熟悉一门数据库也是好的，知己知彼

2019-08-12

Loubobooo 0

我的想法是，可以利用子查询去减少回表操作，既然有主键自增id，便可以利用聚簇索引的优势来强制走索引。代码方法如下：select * from order_detail where id in (select id from order_detail where order_id between 5000 and 10000)

2019-08-11

作者回复

思路是对的，这种方式可以解决。

2019-08-12

新世界 0

查询用order_id索引，然后进行回表查询

2019-08-10

作者回复

对的

2019-08-12

许童童 0

请问该查询选择的索引是什么？有什么方式可以强制使用我们期望的索引呢？
会直接查询主键索引进行全表扫描，因为数据库优化器在判断SQL语句执行使用哪个索引时会计算代价，如果使用主订单 id 索引回表太多，代价太大，还不如用主键索引进行全表扫描。
我觉得SQL改为以下方式可以使用主订单 id 索引，并提高查询效率。
select * from order_detail where id in (select id from order_detail where order_id between 5000 and 10000)

2019-08-10

作者回复

正解！

2019-08-12

undifined 0

可以用子查询实现，因为order_id 的索引中有主键id，先使用order_id 的索引查询到主键id，然后通过主键再从表里查询
SELECT *
FROM order_detail

WHERE id IN (SELECT id FROM order_detail WHERE order_id BETWEEN 5000 AND 10000)
;

2019-08-10

作者回复

对的，思路就是强制使用辅助索引

2019-08-12

-W.LI-

0

索引失效:这个操作在索引树上满足不了需求。就会导致索引失效，b+树的特性就是有序(最左可以理解为高位有效)。
课后习题，感觉老师这么问肯定不会走主订单id的索引了，原因待老师解答。
如果有分布式id生成系统能保证生成的id有序递增的话。可以不用自增的id做主键，直接用生成的oreder_id做主键，可少一次回表。愚见期待老师的答案

2019-08-10

作者回复

已经给出答案，请转至QQ怪的问答题

2019-08-12