09 | 普通索引和唯一索引，应该怎么选择？

2018-12-03 林晓斌

09 | 普通索引和唯一索引，应该怎...
朗读人：林晓斌 15′37″ | 14.32M

今天的正文开始前，我要特意感谢一下评论区几位留下高质量留言的同学。

用户名是 @某、人 的同学，对文章的知识点做了梳理，然后提了关于事务可见性的问题，就是先启动但是后提交的事务，对数据可见性的影响。@夏日雨同学也提到了这个问题，我在置顶评论中回复了，今天的文章末尾也会再展开说明。@Justin 和 @倪大人两位同学提了两个好问题。

对于能够引发更深一步思考的问题，我会在回复的内容中写上“好问题”三个字，方便你搜索，你也可以去看看他们的留言。

非常感谢大家很细致地看文章，并且留下了那么多和很高质量的留言。知道文章有给大家带来一些新理解，对我来说是一个很好的鼓励。同时，也让其他认真看评论区的同学，有机会发现一些自己还没有意识到的、但可能还不清晰的知识点，这也在总体上提高了整个专栏的质量。再次谢谢你们。

好了，现在就回到我们今天的正文内容。

在前面的基础篇文章中，我给你介绍过索引的基本

概念，相信你已经了解了唯一索引和普通索引的区别。今天我们就继续来谈谈，在不同的业务场景下，应该选择普通索引，还是唯一索引？

假设你在维护一个市民系统，每个人都有一个唯一的身份证号，而且业务代码已经保证了不会写入两个重复的身份证号。如果市民系统需要按照身份证号查姓名，就会执行类似这样的 SQL 语句：

```
select name from CUser where id_card = 'xxxxx
```

所以，你一定会考虑在 id_card 字段上建索引。

由于身份证号字段比较大，我不建议你把身份证号当做主键，那么现在你有两个选择，要么给 id_card 字段创建唯一索引，要么创建一个普通索引。如果业务代码已经保证了不会写入重复的身份证号，那么这两个选择逻辑上都是正确的。

现在我要问你的是，从性能的角度考虑，你选择唯一索引还是普通索引呢？选择的依据是什么呢？

简单起见，我们还是用第 4 篇文章《深入浅出索引（上）》中的例子来说明，假设字段 k 上的值都不重复。

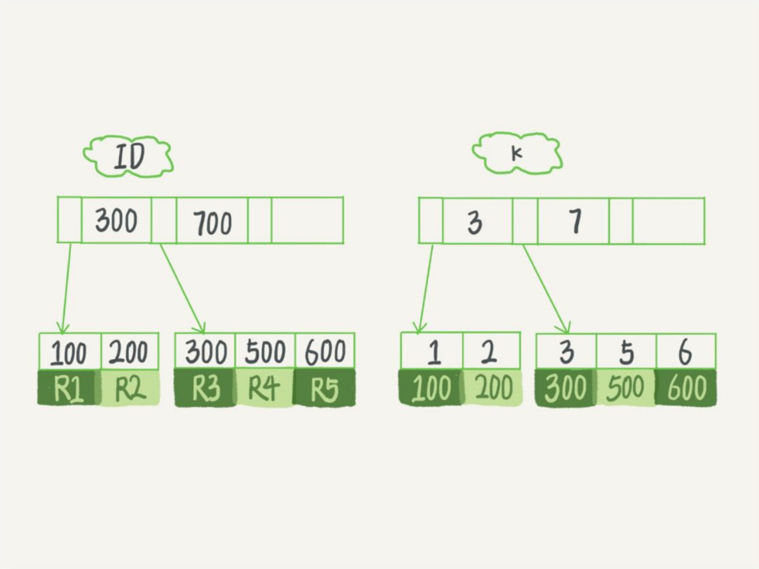


图 1 InnoDB 的索引组织结构

接下来，我们就从这两种索引对查询语句和更新语句的性能影响来进行分析。

查询过程

假设，执行查询的语句是 select id from T where

k=5。这个查询语句在索引树上查找的过程，先是通过 B+ 树从树根开始，按层搜索到叶子节点，也就是图中右下角的这个数据页，然后可以认为数据页内部通过二分法来定位记录。

- 对于普通索引来说，查找到满足条件的第一个记录 (5,500) 后，需要查找下一个记录，直到碰到第一个不满足 k=5 条件的记录。
- 对于唯一索引来说，由于索引定义了唯一性，查找到第一个满足条件的记录后，就会停止继续检索。

那么，这个不同带来的性能差距会有多少呢？答案是，微乎其微。

你知道的，InnoDB 的数据是按数据页为单位来读写的。也就是说，当需要读一条记录的时候，并不是将这个记录本身从磁盘读出来，而是以页为单位，将其整体读入内存。在 InnoDB 中，每个数据页的大小默认是 16KB。

因为引擎是按页读写的，所以说，当找到 k=5 的记

录的时候，它所在的数据页就都在内存里了。那么，对于普通索引来说，要多做的那一次“查找和判断下一条记录”的操作，就只需要一次指针寻找和一次计算。

当然，如果 k=5 这个记录刚好是这个数据页的最后一个记录，那么要取下一个记录，必须读取下一个数据页，这个操作会稍微复杂一些。

但是，我们之前计算过，对于整型字段，一个数据页可以放近千个 key，因此出现这种情况的概率会很低。所以，我们计算平均性能差异时，仍可以认为这个操作成本对于现在的 CPU 来说可以忽略不计。

更新过程

为了说明普通索引和唯一索引对更新语句性能的影响这个问题，我需要先跟你介绍一下 change buffer。

当需要更新一个数据页时，如果数据页在内存中就直接更新，而如果这个数据页还没有在内存中的

话，在不影响数据一致性的前提下，InooDB 会将这些更新操作缓存在 change buffer 中，这样就不需要从磁盘中读入这个数据页了。在下次查询需要访问这个数据页的时候，将数据页读入内存，然后执行 change buffer 中与这个页有关的操作。通过这种方式就能保证这个数据逻辑的正确性。

需要说明的是，虽然名字叫作 change buffer，实际上它是可以持久化的数据。也就是说，change buffer 在内存中有拷贝，也会被写入到磁盘上。

将 change buffer 中的操作应用到原数据页，得到最新结果的过程称为 purge。除了访问这个数据页会触发 purge 外，系统有后台线程会定期 purge。在数据库正常关闭（shutdown）的过程中，也会执行 purge 操作。

显然，如果能够将更新操作先记录在 change buffer，减少读磁盘，语句的执行速度会得到明显的提升。而且，数据读入内存是需要占用 buffer pool 的，所以这种方式还能够避免占用内存，提高内存利用率。

那么，**什么条件下可以使用 change buffer 呢?**

对于唯一索引来说，所有的更新操作都要先判断这个操作是否违反唯一性约束。比如，要插入 (4,400) 这个记录，就要先判断现在表中是否已经存在 k=4 的记录，而这必须要将数据页读入内存才能判断。如果都已经读入到内存了，那直接更新内存会更快，就没必要使用 change buffer 了。

因此，唯一索引的更新就不能使用 change buffer，实际上也只有普通索引可以使用。

change buffer 用的是 buffer pool 里的内存，因此不能无限增大。change buffer 的大小，可以通过参数 innodb_change_buffer_max_size 来动态设置。这个参数设置为 50 的时候，表示 change buffer 的大小最多只能占用 buffer pool 的 50%。

现在，你已经理解了 change buffer 的机制，那么我们再一起来看看**如果要在这张表中插入一个新记录 (4,400) 的话，InnoDB 的处理流程是怎样的。**

第一种情况是，**这个记录要更新的目标页在内存中**。这时，InnoDB 的处理流程如下：

- 对于唯一索引来说，找到 3 和 5 之间的位置，判断到没有冲突，插入这个值，语句执行结束；
- 对于普通索引来说，找到 3 和 5 之间的位置，插入这个值，语句执行结束。

这样看来，普通索引和唯一索引对更新语句性能影响的差别，只是一个判断，只会耗费微小的 CPU 时间。

但，这不是我们关注的重点。

第二种情况是，**这个记录要更新的目标页不在内存中**。这时，InnoDB 的处理流程如下：

- 对于唯一索引来说，需要将数据页读入内存，判断到没有冲突，插入这个值，语句执行结束；
- 对于普通索引来说，则是将更新记录在 change buffer，语句执行就结束了。

将数据从磁盘读入内存涉及随机 IO 的访问，是数

据库里面成本最高的操作之一。change buffer 因为减少了随机磁盘访问，所以对更新性能的提升是会很明显的。

之前我就碰到过一件事儿，有个 DBA 的同学跟我反馈说，他负责的某个业务的库内存命中率突然从 99% 降低到了 75%，整个系统处于阻塞状态，更新语句全部堵住。而探究其原因后，我发现这个业务有大量插入数据的操作，而他在前一天把其中的某个普通索引改成了唯一索引。

change buffer 的使用场景

通过上面的分析，你已经清楚了使用 change buffer 对更新过程的加速作用，也清楚了 change buffer 只限于用在普通索引的场景下，而不适用于唯一索引。那么，现在有一个问题就是：普通索引的所有场景，使用 change buffer 都可以起到加速作用吗?

因为 purge 的时候是真正进行数据更新的时刻，而 change buffer 的主要目的就是将记录的变更动作

缓存下来，所以在一个数据页做 purge 之前，change buffer 记录的变更越多（也就是这个页面上要更新的次数越多），收益就越大。

因此，对于写多读少的业务来说，页面在写完以后马上被访问到的概率比较小，此时 change buffer 的使用效果最好。这种业务模型常见的就是账单类、日志类的系统。

反过来，假设一个业务的更新模式是写入之后马上会做查询，那么即使满足了条件，将更新先记录在 change buffer，但之后由于马上要访问这个数据页，会立即触发 purge 过程。这样随机访问 IO 的次数不会减少，反而增加了 change buffer 的维护代价。所以，对于这种业务模式来说，change buffer 反而起到了副作用。

索引选择和实践

回到我们文章开头的问题，普通索引和唯一索引应该怎么选择。其实，这两类索引在查询能力上是没差别的，主要考虑的是对更新性能的影响。所以，我建议你尽量选择普通索引。

如果所有的更新后面，都马上伴随着对这个记录的查询，那么你应该关闭 change buffer。而在其他情况下，change buffer 都能提升更新性能。

在实际使用中，你会发现，普通索引和 change buffer 的配合使用，对于数据量大的表的更新优化还是很明显的。

特别地，在使用机械硬盘时，change buffer 这个机制的收效是非常显著的。所以，当你有一个类似“历史数据”的库，并且出于成本考虑用的是机械硬盘时，那你应该特别关注这些表里的索引，尽量使用普通索引，然后把 change buffer 尽量开大，以确保这个“历史数据”表的数据写入速度。

change buffer 和 redo log

理解了 change buffer 的原理，你可能会联想到我在前面文章中和你介绍过的 redo log 和 WAL。

在前面文章的评论中，我发现有同学混淆了 redo log 和 change buffer。WAL 提升性能的核心机

制，也的确是尽量减少随机读写，这两个概念确实容易混淆。所以，这里我把它们放到了同一个流程里来说明，便于你区分这两个概念。

> 备注：这里，你可以再回顾下第 2 篇文章《日志系统：一条 SQL 更新语句是如何执行的？》中的相关内容。

现在，我们要在表上执行这个插入语句：

```
mysql> insert into t(id,k) values(id1,k1),(id
```

这里，我们假设当前 k 索引树的状态，查找到位置后，k1 所在的数据页在内存 (InnoDB buffer pool) 中，k2 所在的数据页不在内存中。如图 2 所示是带 change buffer 的更新状态图。

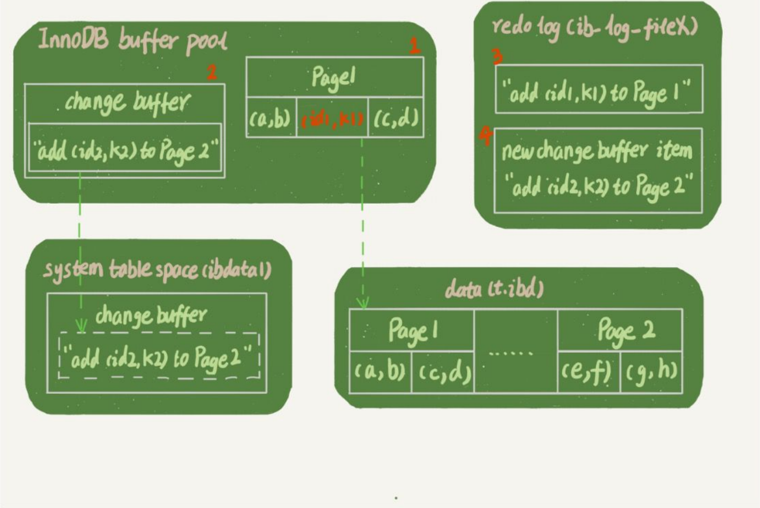


图 2 带 change buffer 的更新过程

分析这条更新语句，你会发现它涉及了四个部分：内存、redo log（ib_log_fileX）、 数据表空间（t.ibd）、系统表空间（ibdata1）。

这条更新语句做了如下的操作（按照图中的数字顺序）：

1. Page 1 在内存中，直接更新内存；
2. Page 2 没有在内存中，就在内存的 change buffer 区域，记录下“我要往 Page 2 插入一

行”这个信息

3. 将上述两个动作记入 redo log 中（图中 3 和 4）。

做完上面这些，事务就可以完成了。所以，你会看到，执行这条更新语句的成本很低，就是写了两处内存，然后写了一处磁盘（两次操作合在一起写了一次磁盘），而且还是顺序写的。

同时，图中的两个虚线箭头，是后台操作，不影响更新的响应时间。

那在这之后的读请求，要怎么处理呢?

比如，我们现在要执行 select * from t where k in (k1, k2)。这里，我画了这两个读请求的流程图。

如果读语句发生在更新语句后不久，内存中的数据都还在，那么此时的这两个读操作就与系统表空间（ibdata1）和 redo log（ib_log_fileX）无关了。所以，我在图中就没画出这两部分。

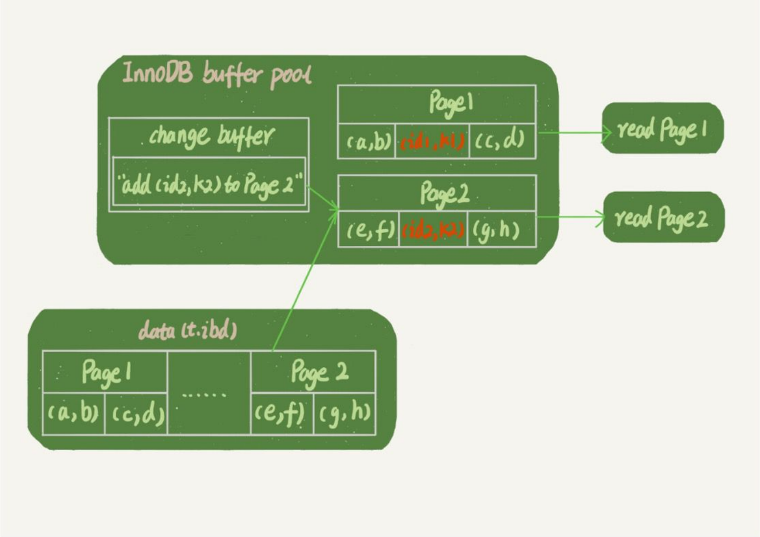


图 3 带 change buffer 的读过程

从图中可以看到：

1. 读 Page 1 的时候，直接从内存返回。有几位同学在前面文章的评论中问到，WAL 之后如果读数据，是不是一定要读盘，是不是一定要从 redo log 里面把数据更新以后才可以返回？其实是不用的。你可以看一下图 3 的这个状态，虽然磁盘上还是之前的数据，但是这里直接从内存返回结果，结果是正确的。

2. 要读 Page 2 的时候，需要把 Page 2 从磁盘读入内存中，然后应用 change buffer 里面的操作日志，生成一个正确的版本并返回结果。

可以看到，直到需要读 Page 2 的时候，这个数据页才会被读入内存。

所以，如果要简单地对比这两个机制在提升更新性能上的收益的话，**redo log 主要节省的是随机写磁盘的 IO 消耗（转成顺序写），而 change buffer 主要节省的则是随机读磁盘的 IO 消耗。**

小结

今天，我从普通索引和唯一索引的选择开始，和你分享了数据的查询和更新过程，然后说明了 change buffer 的机制以及应用场景，最后讲到了索引选择的实践。

由于唯一索引用不上 change buffer 的优化机制，因此如果业务可以接受，从性能角度出发我建议你优先考虑非唯一索引。

最后，又到了思考题时间。

通过图 2 你可以看到，change buffer 一开始是写内存的，那么如果这个时候机器掉电重启，会不会导致 change buffer 丢失呢？change buffer 丢失可不是小事儿，再从磁盘读入数据可就没有了 purge 过程，就等于是数据丢失了。会不会出现这种情况呢？

你可以把你的思考和观点写在留言区里，我会在下一篇文章的末尾和你讨论这个问题。感谢你的收听，也欢迎你把这篇文章分享给更多的朋友一起阅读。

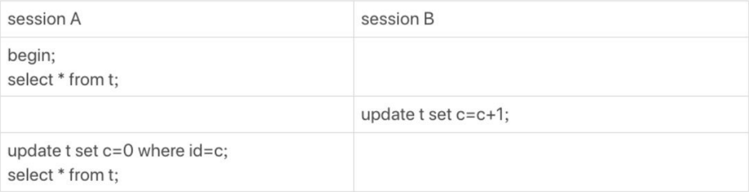
补充：
评论区大家对“是否使用唯一索引”有比较多的讨论，主要是纠结在“业务可能无法确保”的情况。这里，我再说明一下：

- 首先，业务正确性优先。咱们这篇文章的前提是“业务代码已经保证不会写入重复数据”的情况下，讨论性能问题。如果业务不能保证，或者业

- 务就是要求数据库来做约束，那么没得选，必须创建唯一索引。这种情况下，本篇文章的意义在于，如果碰上了大量插入数据慢、内存命中率低的时候，可以给你多提供一个排查思路。
- 然后，在一些“归档库”的场景，你是可以考虑使用唯一索引的。比如，线上数据只需要保留半年，然后历史数据保存在归档库。这时候，归档数据已经是确保没有唯一键冲突了。要提高归档效率，可以考虑把表里面的唯一索引改成普通索引。

上期问题时间

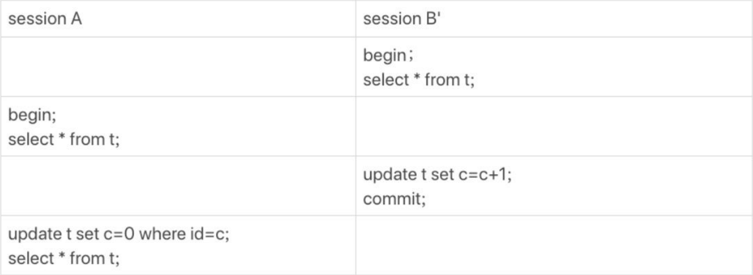
上期的问题是：如何构造一个“数据无法修改”的场景。评论区里已经有不少同学给出了正确答案，这里我再描述一下。



session A	session B
begin; select * from t;	
	update t set c=c+1;
update t set c=0 where id=c; select * from t;	

这样，session A 看到的就是我截图的效果了。

其实，还有另外一种场景，同学们在留言区都还没有提到。



session A	session B'
	begin； select * from t;
begin; select * from t;	
	update t set c=c+1; commit;
update t set c=0 where id=c; select * from t;	

这个操作序列跑出来，session A 看的内容也是能够复现我截图的效果的。这个 session B’启动的事务比 A 要早，其实是上期我们描述事务版本的可见性规则时留的彩蛋，因为规则里还有一个“活跃事务的判断”，我是准备留到这里再补充的。

当我试图在这里讲述完整规则的时候，发现第 8 篇文章《事务到底是隔离的还是不隔离的？》中的解释引入了太多的概念，以致于分析起来非常复杂。

因此，我重写了第 8 篇，这样我们人工去判断可见性的时候，才会更方便。【看到这里，我建议你能

够再重新打开第 8 篇文章并认真学习一次。如果学习的过程中，有任何问题，也欢迎你给我留言】

用新的方式来分析 session B’的更新为什么对 session A 不可见就是：在 session A 视图数组创建的瞬间，session B’是活跃的，属于“版本未提交，不可见”这种情况。

业务中如果要绕过这类问题，@约书亚提供了一个“乐观锁”的解法，大家可以去上一篇的留言区看一下。

评论区留言点赞板：

> @某、人、@夏日雨、@周巘、@李金刚 等同学提了一个很好的问题，就是我们今天答案的 session B’ 的情况；
> @justin 提到了提交和未提交版本的区别对待，@倪大人 提到了读提交和当前读的区别，都是经过了思考后提出的好问题，大家可以去留言区看看。

极客时间
MySQL 实战 45 讲
从原理到实战，丁奇带你搞懂 MySQL
林晓斌
网名丁奇
前阿里资深技术专家

©版权归极客邦科技所有，未经许可不得转载

精选留言

约书亚 0

早，请您看看我以下疑问：

1. 看完后感觉牵扯到之前的内容，又糊涂了。change buffer相当于推迟了更新操作，那对并发控制相关的是否有影响，比如加锁？我一直以为加锁需要把具体的数据页读到内存中来，才能加锁，然而并不是？
2. 在change buffer中有此行记录的情况下，再次更改，是增加一条还是原地修改？
3. purge行为之后应该不会再产生redo log了吧

?

从应用开发的角度看，还是由数据库保证唯一好。

2018-12-03

作者回复

1 3 好问题

1. 锁是一个单独的数据结构，如果数据页上有锁，change buffer 在判断“是否能用”的时候，就会认为否

2. 增加

3. 是这样的，这个问题你分成两步来考虑。
第一步，purge其实是从磁盘读数据页到内存，然后应用，这一步都是更新的内存，没有写文件

现在内存变成脏页了，跟磁盘数据不一样。之后就走刷脏页的流程。刷脏页也不用写。
所以合起来看，就是不用哦

=====

是否使用唯一索引，这个要看业务有没有保证，和性能是否有问题。

有几位同学都提了，我加到文末说明一下。

2018-12-03

虚爱凯平 6

有点疑惑: 主键id也是唯一索引吧? 那我们的新增操作如何利用 change buffer呢?

2018-12-03

作者回复

所以主键索引用不上，都是对于那些二级索引的才有效。

一个insert语句要操作所有索引的嘛，收益在二级索引

2018-12-03

永光 5

会导致change buffer丢失，会导致本次未完成

的操作数据丢失，但不会导致已完成操作的数据丢失。
1.change buffer中分两部分，一部分是本次写入未写完的，一部分是已经写入完成的。
2.针对未写完的，此部分操作，还未写入redo log，因此事务还未提交，所以没影响。
2.针对，已经写完成的，可以通过redo log来进行恢复。

所以，不会对数据库造成影响。

2018-12-03

作者回复

优秀

2018-12-03

峰 2

偶回复下凯平的问题，主键id是主索引，在插入的时候，是根据它来寻找插入位置的，所以已经考虑了它的唯一性。
然后思考题的话，毕竟与 redo写入changebuffer的变化的操作组成了事务，所以不会发生丢失。

2018-12-03

ch 0

两点思考：
1. redo log是写磁盘，直接写数据到磁盘也是写磁盘，这2个效率不一样就是因为redo log是顺序写，直接写数据到磁盘是随机写吗?
2. change buffer里面的内容还没有写入到redo log里面的情况下，再来一个更新操作，会不会把之前的覆盖掉，这个我认为不会，因为更新是独占锁，同一条记录只有事务提交后才会释放锁，所以更新是串行的，只要事务提交才就会把change buffer的内容写入到redo log中。不知道理解的是否正确?

2018-12-03

虚爱凯平 0

感觉今天这篇问题很严重啊, 首先说一下我是第一次接触 change buffer这个概念, 可能产生了什么误会..
我理解的文中讲述change buffer的作用体现在针对普通索引(非主键的都是二级索引, 二级索引又包括了唯一索引和普通索引)在有数据update操作(不包括insert)的时候,能有减少io写操作的功能, 而且这个操作是提现在更新表数据上的. 为什么我在这里会理解成update操作呢.. (Inno

DB中表就是按索引的方式存放的, 即使我们不主动创建主键 也会生成一个默认的row_id来当做主键, 意味着表一定是有一个主键, 即唯一索引. insert操作 一定会涉及主键索引的变动, 所以change buffer针对 insert 是完全没有用的吗??)

针对change buffer 我百度了一下, 有文章描述change buffer 是针对表中包含普通索引的表在insert操作时, 优化 普通索引的更新(在insert时,不会立即更新普通索引 而是保存到change buffer延迟处理). 这么一个功能. 不知道 这个理解是否正确呢?

2018-12-03

高原

0

ChangeBuffer就是对《日志系统：一条 SQL 更新语句是如何执行的？》中 update 语句执行流程进一步优化。更新步骤：先更新内存，再通过redo log更新磁盘。

而Change Buffer的引入使得更新内存都是按需更新，只有在需要读的时候才更新。 如果这样理解对的话，就有一个问题，如果在purge之前，redo log执行了磁盘写，那change buffer里

的操作是不是就有冲突怎么让失效呢?

2018-12-03

蠢蠢欲动的腹肌 0

老师，你好
想问下purge与redolog的commit的关系或者执行顺序是怎样的，感觉这里比较模糊

2018-12-03

岁月安然 0

之前第二章说redo-log是优先记录的，想问下
1、charge-buffer是在什么时候持久化写入系统表空间的?
2、既然redo-log也记录了charge-buffer的操作，为什么还需要写入系统表空间?

2018-12-03

肖鹏 0

数据的持久性是由redo保证的，断电后的恢复是由redo和扫最近的binlog完成的。和change buffer无关。另外唯一索引会是锁变得非常复杂吧，会有gap lock?

2018-12-03

老杨同志 0

首先感谢一下老师，这门课很有收获。老师很勤奋，留言回复特别多。

试着回答一下思考题：

结论是：change buffer 在断电后不会丢失。原因老师在文中已经写了。写change buffer的动作也被写入了redo log，系统重新启动的时候执行redo log会恢复断电前的change buffer

2018-12-03

憶海拾貝 0

唯一字段不加唯一索引, 墨菲定律会出来搞事. --记一次踩坑经验..

2018-12-03

作者回复

嗯这里其实要说等是“业务要保证”

当要性能问题的时候，多一个check的点

2018-12-03

张立春 0

老师好！mysql数据库是码农无法绕过的知识，第一时间看到有这个课就定下了，看到有两万

多人订购，就知道有多火爆了。已经跟学几节课，总的感觉方向和内容都挺好，但我自己感觉在细节和完备性上可以再深入打磨一下，也许和我自身水平有关。有些知识如果觉得太小儿科，可以只给出参考链接，但最好不要缺失。

比如本次课，讲解一般索引和普通索引，列举查询和更新两种操作的差异。在查询上，我的疑问是table schema是什么？建表建库参数是什么？table只包括普通索引或唯一索引或两者都有的情况下是否有区别? 是否只是根据查询所依赖的条件字段?

关于更新，有增删改三种操作，所举例子只有增加，其它两种有什么差异? 如果两种索引都存在表中是否不同?

我建议每次课都把上下文环境写清楚：mysql版本/建表建库命令？连接参数？数据库配置参数？目的是大家能够根据这个自己动手重现和实际感受所讲的知识，我建议你建立相应的docker和github，把相关环境、脚本和代码放上去，大家也可以自由修改和发表想法。talk is cheap，show me code。看懂–理解–会用–解决问题，每一步都有很大差异，我希望学了这门课可以真正用在实际工作中去解决问题。

我相信这门课完结之时，大家受益匪浅，但可能作者在认知/知识上的收益却是最大的。因为肩负两万多人的信任和期望，这份压力和动力可能是绝无仅有的，对个人的锻炼也是无法用言语表达的。

2018-12-03

作者回复

你说得很对，如果是有可观测效果的，我会尽量放例子（复现步骤）哈

比如怎么复现锁等待啊、优化器啊这些的，

今天这会话题比较特殊，也是我确实没有有意设计，后面会尽量给出复现步骤。（类似08篇
）

12

写留言

请朋友读