9 普通索引和唯一索引，该怎么选

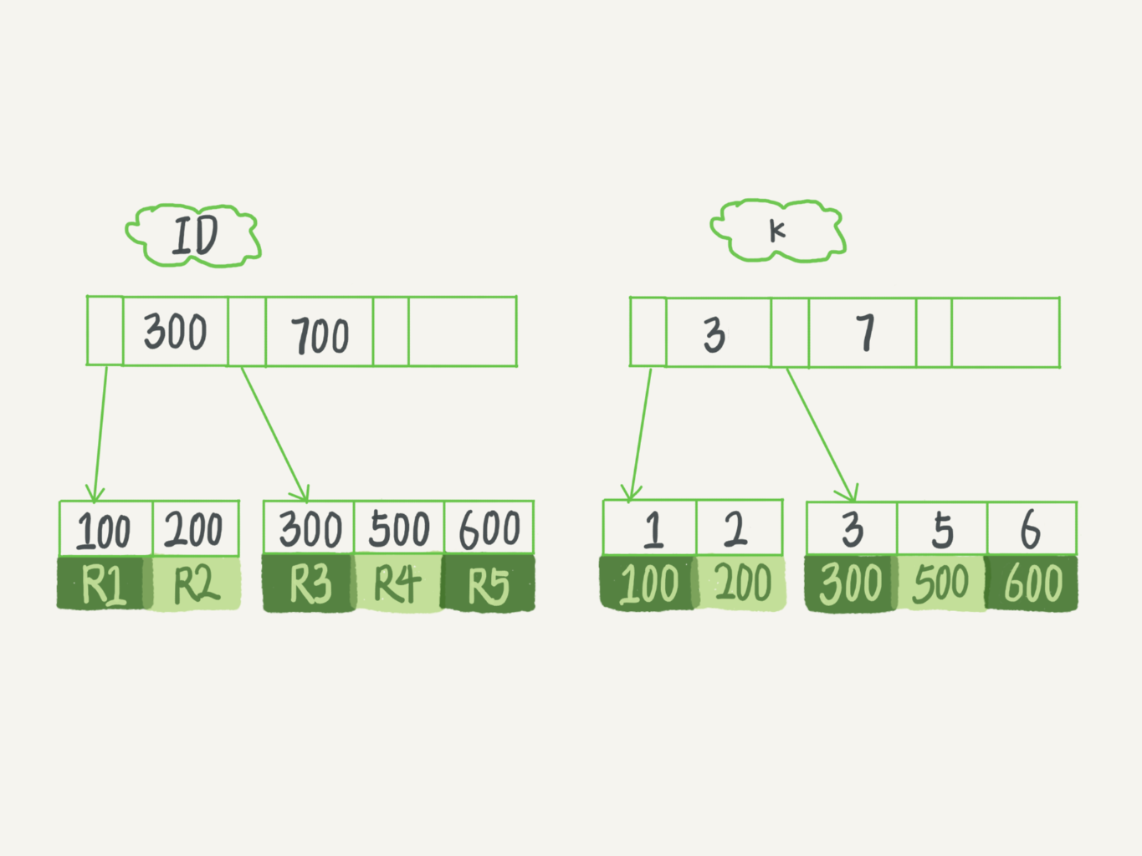
维护一个市民系统，每个人都有一个唯一的身份证号，而且业务代码已经保证了不会写如两个重复的身份证号，如果市民系统需要按照身份证号查姓名，就户执行类似这样的SQL语句：

Select name from CUser where id\_card = ‘xxxxxxyyyyyyzzzzz’;

你一定会考虑在id\_card字段行建索引。

由于身份证号字段比较大，我不建议把身份证号做主键，①要么现在有连个选择，②要么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不会写入重复的身份证号。那么两个选择逻辑上都正确的。

但是，从性能角度考虑，选择唯一索引还是普通索引？选择的依据是什么？



从以前的例子来说明，假设k上的值都不重复。

**查询过程**

执行查询语句是select id from t where k=5 这个查询语句在索引上查找的过程，显示通过B+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通过二分发来定位记录。

1. 对于普通索引，查找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
2. 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异会有多少呢？答案是：微乎其微

InnoDB,的数据是按页为单位读取来的，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，每一个数据页的大小默认是16kb,

引擎是按页读写的，所以找到k=5的记录时，它所在的数据页就都在内存里了，那么对于普通索引，要多做一次“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，那么要区下一条，记录必须读取下一个数据页，这个操作会稍微复杂一些。

之前计算过，对于整型字段，一个数据页可以放仅千个key,因此出现这种情况的概率会很低，计算平均性能差异时，仍可认为这个操作对于现在的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，需要先介绍一下change buffer.

当更新一个数据页时，如果数据也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，InnoDb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，在下次需要查询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，虽然名字叫做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merge,除了访问这个数据页会触发merge外，系统后台线程会定期merge,在数据库正常关闭（shutdown）的过程中，也会执行merge操作。

如果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buffer pool的，所以这中方式还是能够避免占用内存，提升效率。

那么，**什么条件下可以使用change buffer呢？**

对于唯一索引来说，所有的更新操纵要先判断这个操作是否违反唯一行约束，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，如果都已经读入的内存了，那直接更新内存会更快，就没必要使用change buffer.

因此，唯一索引的更新就不能使用change buffer,实际上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，表示change buffer的大小最多只占用buffer pool的50%。

**如果要在这张表中插入一个新纪录（4，400）的话，InnoDB的处理流程是怎样的。**

**第一种情况，这个记录更新的目标页在内存中**，innoDB处理流程如下

1. 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
2. 对于普通索引来说，找打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。

**第二种情况时：这个记录更新的目标不在内存中，**这时，InnoDB的处理流程

1. 对于唯一索引来说，需要将数据页读入到内存，判断到没有冲突，插入这个值，语句执行结束
2. 对于普通索引来说，则是将更新记录在change buffer，语句执行结束

将数据从磁盘读入到内存涉及随机IO的访问，是数据库里面成本最高的操作之一，change buffer因为减少随机磁盘访问，所以对更新性能的提升会很明显。

**Chage buffer 使用场景**

通过分析，change buffer 只限于在普通索引的情况下，而不适用于唯一索引，那么普通索引的索引场景，使用change buffer都可以起到加速作用吗？

因为merge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merge之前，change buffer记录的变更越多（也就是这个页面上更新的次数越多），收益越大。

因此，对于写多读少的业务，页面在写完以后马上被访问的概率比较小，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，日之类。

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merge过程，这样随机访问IO的次数不会减少，返而会增加change buffer的维护代价，所以这种业务模式来说，change buffer反而起副作用。

**索引选择和实践**

**这两类索引在查询能力上是马差别的，主要考虑是对更新性能的影响，建议尽量选择普通索引。**

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。

在实际的使用中，普通索引和change buffer的配合使用，对于数据量大的更新优化还是很明显的

特别的，在使用机械硬盘时，change buffer这个机制的是收效是非常显著的，当有一个类似“历史数据”的库，并且处于成本考虑用的是机械硬盘时，应该特别注意这些表里的索引，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

在上表上执行这个插入语句

**mysql> insert into t(id,k) values(id1,k1),(id2,k2);**

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所示带change buffer的更新状态

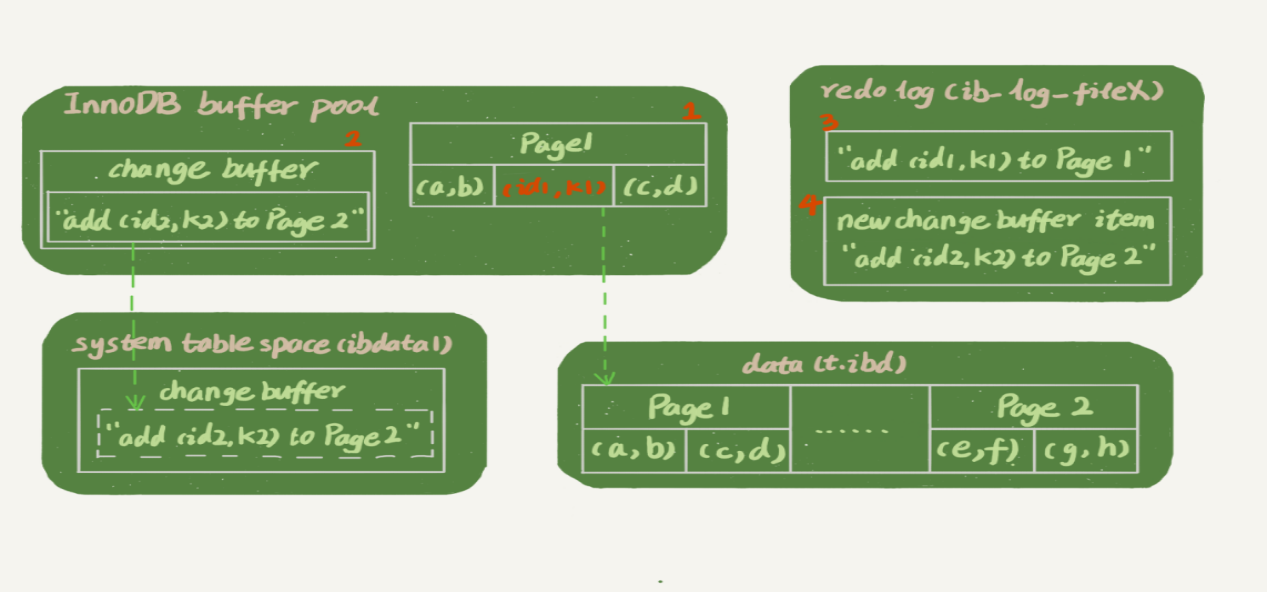


图 2 带 change buffer的更新过程

分析这条更新语句，会发现涉及了4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操作（按图中的数字顺序）

1. Page1 在内存中，直接更新内存；
2. Page2 没有在内存中，就在内存的change buffer区域，记录下“我要往Page2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）

事务就可以完成了，所以，执行这条更新语句的效率很低，就写来两处内存，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。

在之后读请求，要怎么处理呢？

比如，要执行 select \* from t where k in(k1,k2).这里，有两个读流程图

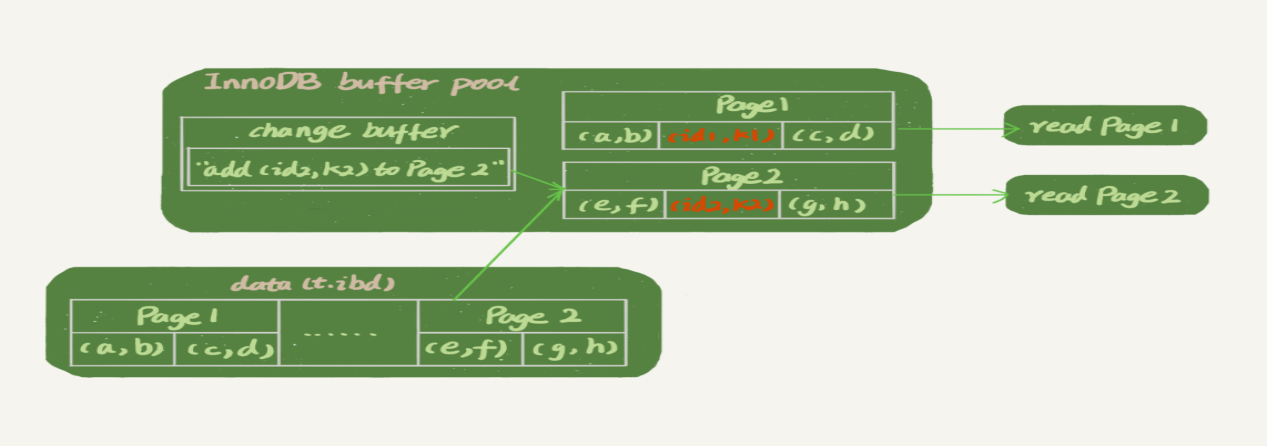
如果读语句发生在更新语句后不久，内存中的数据都还在，那么此时的这两个读操作就与系统表空间（ibdata1）和redo log(ib\_log\_fileX)无关了，所以图中没有了这两部分

图3， 带change buffer的读过程

从图中可以看到

1. 读Page1的时候，直接从内存返回，有疑问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里面吧数据更新以后才可以返回？其实是不用的，可以看图3的状态，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的时候，需要把page2从磁盘读入到内存，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。

直到需要读page2的时候，这个数据页才会被读入内存。

所以，如果简单的对比这个两个机制在提升更新性能的收益的话，**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**

**补充:是否使用唯一索引的情况，主要纠结在“业务可能无法保证”的情况下。**

首先，业务正确性优先，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，如果碰上了大量插入数据慢，内存命中率低的时候，可以提供一个排查的思路

然后，在一些“归档库”的场景，可以考虑使用普通索引，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，此时，归档数据已经是确保没有唯一键冲突了，要提高归档效率，可以考虑吧表里面的唯一索引改成普通索引。

**讨论题**

通过图2 可以看到，change buffer一开始写内存，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， 再从磁盘读入数据可就没有了merge过程，等于数据丢失了，会不会有这种情况呢？

答案：不会丢失数据，虽然只是跟新内存，但是事务提交的时候，我们把change buffer的操作也记录到了redo log里了，所以崩溃恢复的时候，change buffer 也能找回来。

Merge的执行流程

1. 从磁盘读入到数据页到内存（老版本的数据页）
2. 从change buffer 里找出这个数据页的change buffer记录（可能有多个），依次应用，得到新版数据页。
3. 写redolog.这个redo log 包含了数据的变更和change buffer的变更

到这里merge过程就结束了，此时，数据页和内存中change buffer对应的磁盘位置都还没有修改，属于脏页，之后各自刷回自己的物理数据，就是另外一回事了。