MySQL 加锁处理分析

12月 13th, 2013
发表评论 | Trackback

1. 背景

MySQL/InnoDB的加锁分析，一直是一个比较困难的话题。我在工作过程中，经常会有同事咨询这方面的问题。同时，微博上也经常会收到MySQL锁相关的私信，让我帮助解决一些死锁的问题。本文，准备就MySQL/InnoDB的加锁问题，展开较为深入的分析与讨论，主要是介绍一种思路，运用此思路，拿到任何一条SQL语句，都能完整的分析出这条语句会加什么锁？会有什么样的使用风险？甚至是分析线上的一个死锁场景，了解死锁产生的原因。

注：MySQL是一个支持插件式存储引擎的数据库系统。本文下面的所有介绍，都是基于InnoDB存储引擎，其他引擎的表现，会有较大的区别。

1. MVCC：Snapshot Read vs Current Read

MySQL InnoDB存储引擎，实现的是基于多版本的并发控制协议——MVCC (Multi-Version Concurrency Control) (注：与MVCC相对的，是基于锁的并发控制，Lock-Based Concurrency Control)。MVCC最大的好处，相信也是耳熟能详：读不加锁，读写不冲突。在读多写少的OLTP应用中，读写不冲突是非常重要的，极大的增加了系统的并发性能，这也是为什么现阶段，几乎所有的RDBMS，都支持了MVCC。

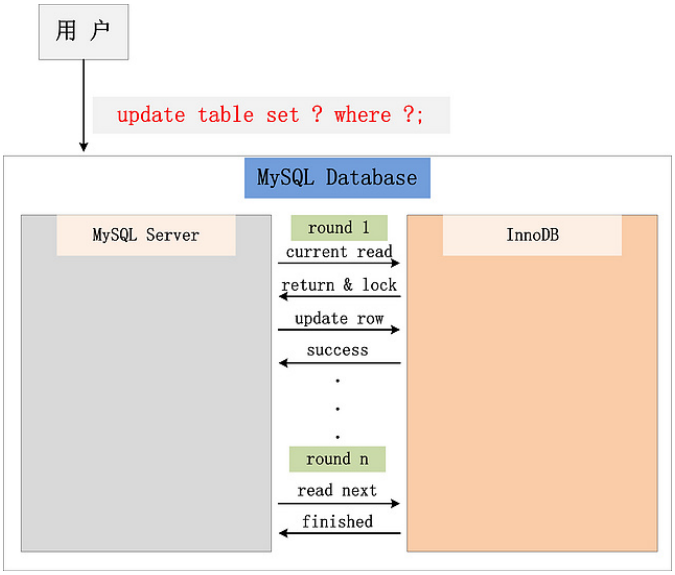
在MVCC并发控制中，读操作可以分成两类：快照读 (snapshot read)与当前读 (current read)。快照读，读取的是记录的可见版本 (有可能是历史版本)，不用加锁。当前读，读取的是记录的最新版本，并且，当前读返回的记录，都会加上锁，保证其他事务不会再并发修改这条记录。

在一个支持MVCC并发控制的系统中，哪些读操作是快照读？哪些操作又是当前读呢？以MySQL InnoDB为例：

- **快照读：**简单的select操作，属于快照读，不加锁。(当然，也有例外，下面会分析)
  - select * from table where ?;
- **当前读：**特殊的读操作，插入/更新/删除操作，属于当前读，需要加锁。
  - select * from table where ? lock in share mode;
  - select * from table where ? for update;
  - insert into table values (...);
  - update table set ? where ?;
  - delete from table where ?;

  所有以上的语句，都属于当前读，读取记录的最新版本。并且，读取之后，还需要保证其他并发事务不能修改当前记录，对读取记录加锁。其中，除了第一条语句，对读取记录加S锁 (共享锁)外，其他的操作，都加的是X锁 (排它锁)。

为什么将 插入/更新/删除 操作，都归为当前读？可以看看下面这个 更新 操作，在数据库中的执行流程：



从图中，可以看到，一个Update操作的具体流程。当Update SQL被发给MySQL后，MySQL Server会根据where条件，读取第一条满足条件的记录，然后InnoDB引擎会将第一条记录返回，并加锁 (current read)。待MySQL Server收到这条加锁的记录之后，会再发起一个Update请求，更新这条记录。一条记录操作完成，再读取下一条记录，直至没有满足条件的记录为止。因此，Update操作内部，就包含了一个当前读。同理，Delete操作也一样。Insert操作会稍微有些不同，简单来说，就是Insert操作可能会触发Unique Key的冲突检查，也会进行一个当前读。

注：根据上图的交互，针对一条当前读的SQL语句，InnoDB与MySQL Server的交互，是一条一条进行的，因此，加锁也是一条一条进行的。先对一条满足条件的记录加锁，返回给MySQL Server，做一些DML操作；然后在读取下一条加锁，直至读取完毕。

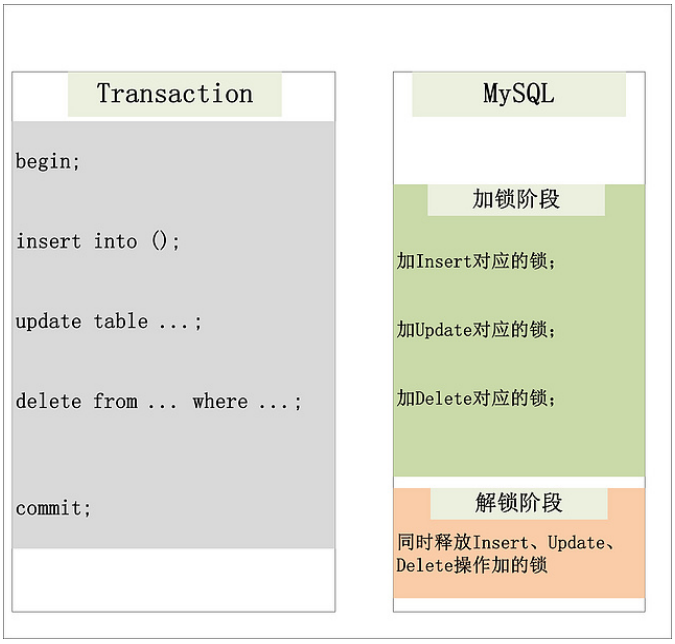
1. Cluster Index：聚簇索引

InnoDB存储引擎的数据组织方式，是聚簇索引表：完整的记录，存储在主键索引中，通过主键索引，就可以获取记录所有的列。关于聚簇索引表的组织方式，可以参考MySQL的官方文档：Clustered and Secondary Indexes 。本文假设读者对这个，已经有了一定的认识，就不再做具体的介绍。接

下来的部分，主键索引/聚簇索引 两个名称，会有一些混用，望读者知晓。

1. 2PL：Two-Phase Locking

传统RDBMS加锁的一个原则，就是2PL (二阶段锁)：Two-Phase Locking。相对而言，2PL比较容易理解，说的是锁操作分为两个阶段：加锁阶段与解锁阶段，并且保证加锁阶段与解锁阶段不相交。下面，仍旧以MySQL为例，来简单看看2PL在MySQL中的实现。



从上图可以看出，2PL就是将加锁/解锁分为两个完全不相交的阶段。加锁阶段：只加锁，不放锁。解锁阶段：只放锁，不加锁。

1. Isolation Level

隔离级别：Isolation Level，也是RDBMS的一个关键特性。相信对数据库有所了解的朋友，对于4种隔离级别：Read Uncommited，Read Committed，Repeatable Read，Serializable，都有了深入的认识。本文不打算讨论数据库理论中，是如何定义这4种隔离级别的含义的，而是跟大家介绍一下MySQL/InnoDB是如何定义这4种隔离级别的。

MySQL/InnoDB定义的4种隔离级别：

- **Read Uncommited**

  可以读取未提交记录。此隔离级别，不会使用，忽略。

- **Read Committed (RC)**

  快照读忽略，本文不考虑。

  针对当前读，**RC隔离级别保证对读取到的记录加锁 (记录锁)**，存在幻读现象。

- **Repeatable Read (RR)**

  快照读忽略，本文不考虑。

  针对当前读，**RR隔离级别保证对读取到的记录加锁 (记录锁)，同时保证对读取的范围加锁，新的满足查询条件的记录不能够插入 (间隙锁)**，不存在幻读现象。

- **Serializable**

  从MVCC并发控制退化为基于锁的并发控制。不区别快照读与当前读，所有的读操作均为当前读，读加读锁 (S锁)，写加写锁 (X锁)。

  Serializable隔离级别下，读写冲突，因此并发度急剧下降，在MySQL/InnoDB下不建议使用。

1. 一条简单SQL的加锁实现分析

在介绍完一些背景知识之后，本文接下来将选择几个有代表性的例子，来详细分析MySQL的加锁处理。当然，还是从最简单的例子说起。经常有朋友发给我一个SQL，然后问我，这个SQL加什么锁？就如同下面两条简单的SQL，他们加什么锁？

- **SQL1：**select * from t1 where id = 10;
- **SQL2：**delete from t1 where id = 10;

针对这个问题，该怎么回答？我能想象到的一个答案是：

- **SQL1：**不加锁。因为MySQL是使用多版本并发控制的，读不加锁。
- **SQL2：**对id = 10的记录加写锁 (走主键索引)。

这个答案对吗？说不上来。即可能是正确的，也有可能是错误的，已知条件不足，这个问题没有答案。如果让我来回答这个问题，我必须还要知道以下的一些前提，前提不同，我能给出的答案也就不同。要回答这个问题，还缺少哪些前提条件？

- **前提一：**id列是不是主键？
- **前提二：**当前系统的隔离级别是什么？
- **前提三：**id列如果不是主键，那么id列上有索引吗？
- **前提四：**id列上如果有二级索引，那么这个索引是唯一索引吗？
- **前提五：**两个SQL的执行计划是什么？索引扫描？全表扫描？

没有这些前提，直接就给定一条SQL，然后问这个SQL会加什么锁，都是很业余的表现。而当这些问题有了明确的答案之后，给定的SQL会加什么锁，也就一目了然。下面，我将这些问题的答案进行组合，然后按照从易到难的顺序，逐个分析每种组合下，对应的SQL会加哪些锁？

注：下面的这些组合，我做了一个前提假设，也就是有索引时，执行计划一定会选择使用索引进行过滤 (索引扫描)。但实际情况会复杂很多，真正的执行计划，还是需要根据MySQL输出的为准。

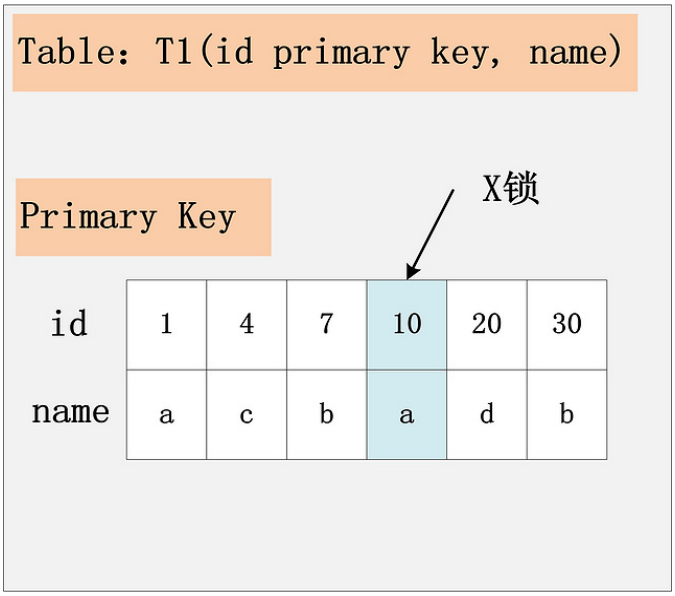
- **组合一：**id列是主键，RC隔离级别
- **组合二：**id列是二级唯一索引，RC隔离级别
- **组合三：**id列是二级非唯一索引，RC隔离级别
- **组合四：**id列上没有索引，RC隔离级别
- **组合五：**id列是主键，RR隔离级别
- **组合六：**id列是二级唯一索引，RR隔离级别
- **组合七：**id列是二级非唯一索引，RR隔离级别
- **组合八：**id列上没有索引，RR隔离级别
- **组合九：**Serializable隔离级别

排列组合还没有列举完全，但是看起来，已经很多了。真的有必要这么复杂吗？事实上，要分析加锁，就是需要这么复杂。但是从另一个角度来说，只要你选定了一种组合，SQL需要加哪些锁，其实也就确定了。接下来，就让我们来逐个分析这9种组合下的SQL加锁策略。

注：在前面八种组合下，也就是RC，RR隔离级别下，SQL1：select操作均不加锁，采用的是快照读，因此在下面的讨论中就忽略了，主要讨论SQL2：delete操作的加锁。

1. 组合一：id主键+RC

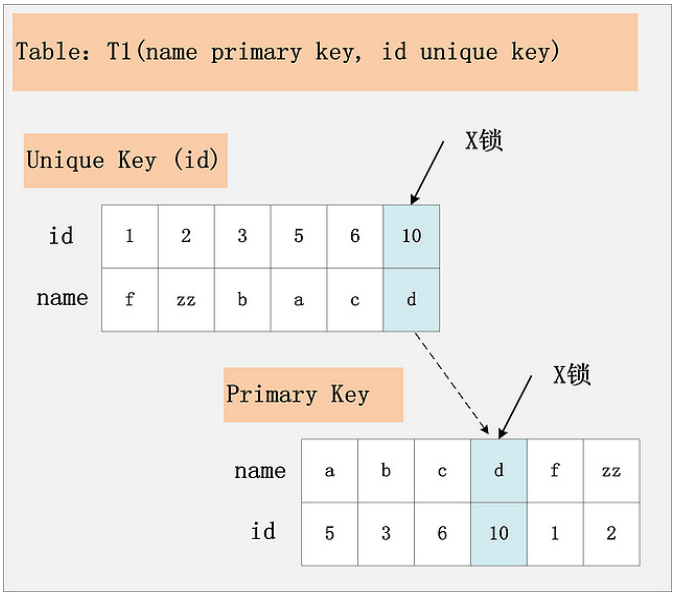
这个组合，是最简单，最容易分析的组合。id是主键，Read Committed隔离级别，给定SQL：delete from t1 where id = 10; 只需要将主键上，id = 10的记录加上X锁即可。如下图所示：



结论：id是主键时，此SQL只需要在id=10这条记录上加X锁即可。

1. 组合二：id唯一索引+RC

这个组合，id不是主键，而是一个Unique的二级索引键值。那么在RC隔离级别下，delete from t1 where id = 10; 需要加什么锁呢？见下图：

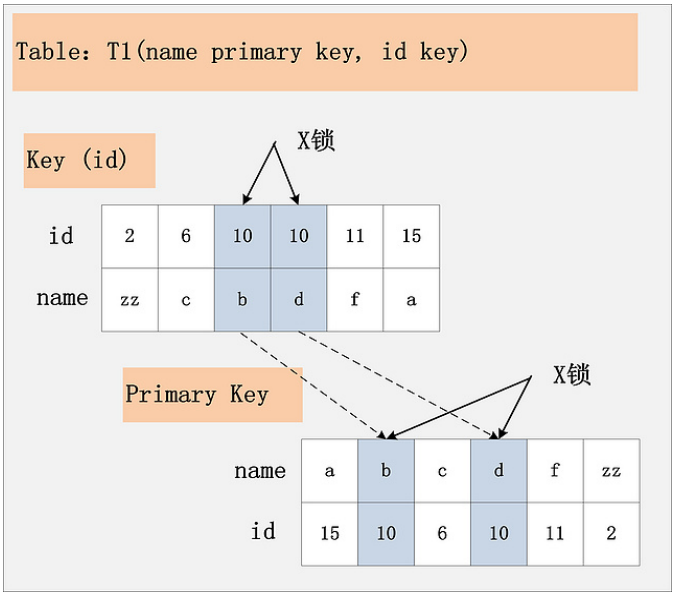


此组合中，id是unique索引，而主键是name列。此时，加锁的情况由于组合一有所不同。由于id是unique索引，因此delete语句会选择走id列的索引进行where条件的过滤，在找到id=10的记录后，首先会将unique索引上的id=10索引记录加上X锁，同时，会根据读取到的name列，回主键索引(聚簇索引)，然后将聚簇索引上的name = ‘d’ 对应的主键索引项加X锁。为什么聚簇索引上的记录也要加锁？试想一下，如果并发的一个SQL，是通过主键索引来更新：update t1 set id = 100 where name = ‘d’；此时，如果delete语句没有将主键索引上的记录加锁，那么并发的update就会感知不到delete语句的存在，违背了同一记录上的更新/删除需要串行执行的约束。

结论：若id列是unique列，其上有unique索引。那么SQL需要加两个X锁，一个对应于id unique索引上的id = 10的记录，另一把锁对应于聚簇索引上的[name=’d’,id=10]的记录。

1. 组合三：id非唯一索引+RC

相对于组合一、二，组合三又发生了变化，隔离级别仍旧是RC不变，但是id列上的约束又降低了，id列不再唯一，只有一个普通的索引。假设delete from t1 where id = 10; 语句，仍旧选择id列上的索引进行过滤where条件，那么此时会持有哪些锁？同样见下图：

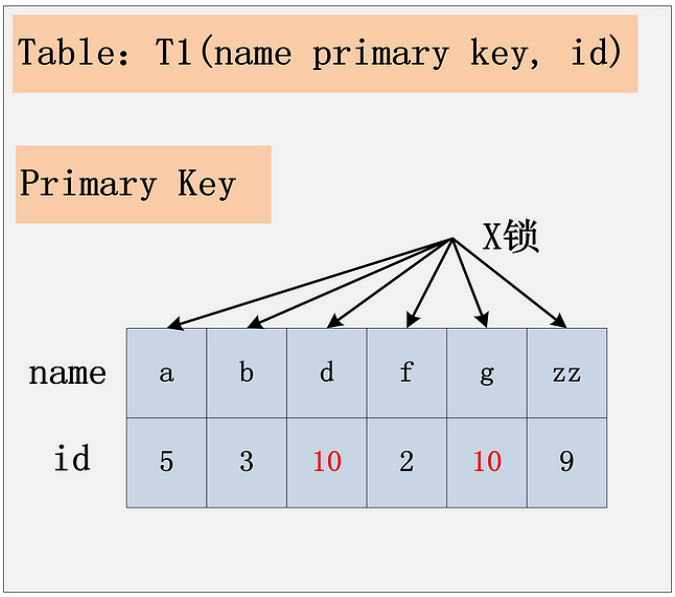


根据此图，可以看到，首先，id列索引上，满足id = 10查询条件的记录，均已加锁。同时，这些记录对应的主键索引上的记录也都加上了锁。与组合二唯一的区别在于，组合二最多只有一个满足等值查询的记录，而组合三会将所有满足查询条件的记录都加锁。

结论：若id列上有非唯一索引，那么对应的所有满足SQL查询条件的记录，都会被加锁。同时，这些记录在主键索引上的记录，也会被加锁。

1. 组合四：id无索引+RC

相对于前面三个组合，这是一个比较特殊的情况。id列上没有索引，where id = 10;这个过滤条件，没法通过索引进行过滤，那么只能走全表扫描做过滤。对应于这个组合，SQL会加什么锁？或者是换句话说，全表扫描时，会加什么锁？这个答案也有很多：有人说会在表上加X锁；有人说会将聚簇索引上，选择出来的id = 10;的记录加上X锁。那么实际情况呢？请看下图：



由于id列上没有索引，因此只能走聚簇索引，进行全部扫描。从图中可以看到，满足删除条件的记录有两条，但是，聚簇索引上所有的记录，都被加上了X锁。无论记录是否满足条件，全部被加上X锁。既不是加表锁，也不是在满足条件的记录上加行锁。

有人可能会问？为什么不是只在满足条件的记录上加锁呢？这是由于MySQL的实现决定的。如果一个条件无法通过索引快速过滤，那么存储引擎层面就会将所有记录加锁后返回，然后由MySQL Server层进行过滤。因此也就把所有的记录，都锁上了。

注：在实际的实现中，MySQL有一些改进，在MySQL Server过滤条件，发现不满足后，会调用unlock_row方法，把不满足条件的记录放锁 (违背了2PL的约束)。这样做，保证了最后只会持有满足条件记录上的锁，但是每条记录的加锁操作还是不能省略的。

结论：若id列上没有索引，SQL会走聚簇索引的全扫描进行过滤，由于过滤是由MySQL Server层面进行的。因此每条记录，无论是否满足条件，都会被加上X锁。但是，为了效率考量，MySQL做了优化，对于不满足条件的记录，会在判断后放锁，最终持有的，是满足条件的记录上的锁，但是不满足条件的记录上的加锁/放锁动作不会省略。同时，优化也违背了2PL的约束。

1. 组合五：id主键+RR

上面的四个组合，都是在Read Committed隔离级别下的加锁行为，接下来的四个组合，是在Repeatable Read隔离级别下的加锁行为。

组合五，id列是主键列，Repeatable Read隔离级别，针对delete from t1 where id = 10; 这条SQL，加锁与组合一：[id主键，Read Committed]一致。

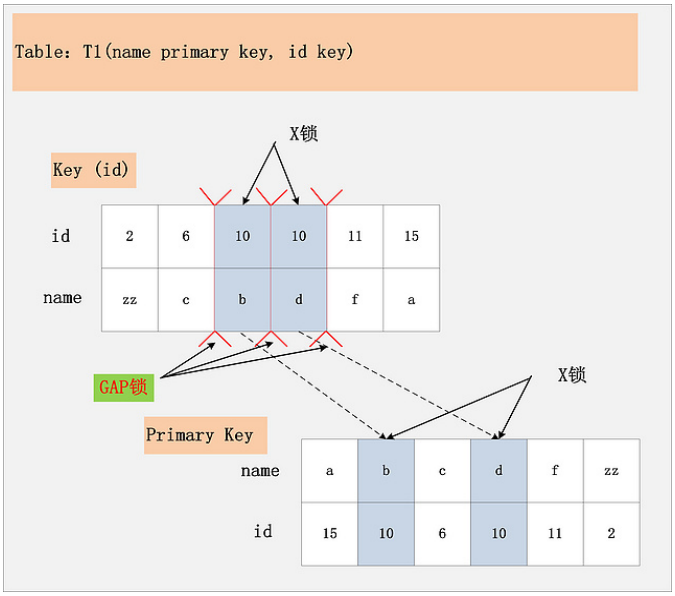
1. 组合六：id唯一索引+RR

与组合五类似，组合六的加锁，与组合二：[id唯一索引，Read Committed]一致。两个X锁，id唯一索引满足条件的记录上一个，对应的聚簇索引上的记录一个。

1. 组合七：id非唯一索引+RR

还记得前面提到的MySQL的四种隔离级别的区别吗？RC隔离级别允许幻读，而RR隔离级别，不允许存在幻读。但是在组合五、组合六中，加锁行为又是与RC下的加锁行为完全一致。那么RR隔离级别下，如何防止幻读呢？问题的答案，就在组合七中揭晓。

组合七，Repeatable Read隔离级别，id上有一个非唯一索引，执行delete from t1 where id = 10; 假设选择id列上的索引进行条件过滤，最后的加锁行为，是怎么样的呢？同样看下面这幅图：



此图，相对于组合三：[id列上非唯一锁，Read Committed]看似相同，其实却有很大的区别。最大的区别在于，这幅图中多了一个GAP锁，而且GAP锁看起来也不是加在记录上的，倒像是加载两条记录之间的位置，GAP锁有何用？

其实这个多出来的GAP锁，就是RR隔离级别，相对于RC隔离级别，不会出现幻读的关键。确实，GAP锁锁住的位置，也不是记录本身，而是两条记录之间的GAP。所谓幻读，就是同一个事务，连续做两次当前读 (例如：select * from t1 where id = 10 for update;)，那么这两次当前读返回的是完全相同的记录 (记录数量一致，记录本身也一致)，第二次的当前读，不会比第一次返回更多的记录 (幻象)。

如何保证两次当前读返回一致的记录，那就需要在第一次当前读与第二次当前读之间，其他的事务不会插入新的满足条件的记录并提交。为了实现这个功能，GAP锁应运而生。

如图中所示，有哪些位置可以插入新的满足条件的项 (id = 10)，考虑到B+树索引的有序性，满足条件的项一定是连续存放的。记录[6,c]之前，不会插入id=10的记录；[6,c]与[10,b]间可以插入[10, aa]；[10,b]与[10,d]间，可以插入新的[10,bb],[10,c]等；[10,d]与[11,f]间可以插入满足条件的[10,e], [10,z]等；而[11,f]之后也不会插入满足条件的记录。因此，为了保证[6,c]与[10,b]间，[10,b]与[10,d]间，[10,d]与[11,f]不会插入新的满足条件的记录，MySQL选择了用GAP锁，将这三个GAP给锁起来。

Insert操作，如insert [10,aa]，首先会定位到[6,c]与[10,b]间，然后在插入前，会检查这个GAP是否已经被锁上，如果被锁上，则Insert不能插入记录。因此，通过第一遍的当前读，不仅将满足条件的记录锁上 (X锁)，与组合三类似。同时还是增加3把GAP锁，将可能插入满足条件记录的3个GAP给锁上，保证后续的Insert不能插入新的id=10的记录，也就杜绝了同一事务的第二次当前读，出现幻象的情况。

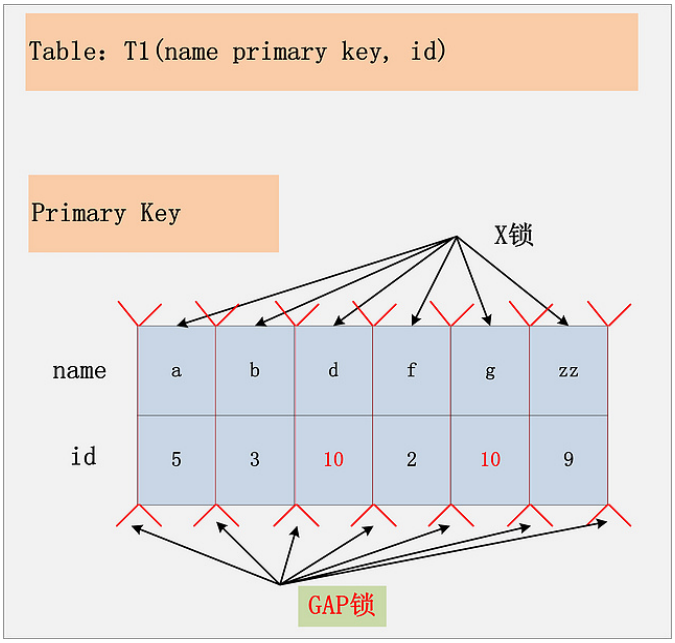
有心的朋友看到这儿，可以会问：既然防止幻读，需要靠GAP锁的保护，为什么组合五、组合六，也是RR隔离级别，却不需要加GAP锁呢？

首先，这是一个好问题。其次，回答这个问题，也很简单。GAP锁的目的，是为了防止同一事务的两次当前读，出现幻读的情况。而组合五，id是主键；组合六，id是unique键，都能够保证唯一性。一个等值查询，最多只能返回一条记录，而且新的相同取值的记录，一定不会在新插入进来，因此也就避免了GAP锁的使用。其实，针对此问题，还有一个更深入的问题：如果组合五、组合六下，针对SQL：select * from t1 where id = 10 for update; 第一次查询，没有找到满足查询条件的记录，那么GAP锁是否还能够省略？此问题留给大家思考。

结论：Repeatable Read隔离级别下，id列上有一个非唯一索引，对应SQL：delete from t1 where id = 10; 首先，通过id索引定位到第一条满足查询条件的记录，加记录上的X锁，加GAP上的GAP锁，然后加主键聚簇索引上的记录X锁，然后返回；然后读取下一条，重复进行。直至进行到第一条不满足条件的记录[11,f]，此时，不需要加记录X锁，但是仍旧需要加GAP锁，最后返回结束。

1. 组合八：id无索引+RR

组合八，Repeatable Read隔离级别下的最后一种情况，id列上没有索引。此时SQL：delete from t1 where id = 10; 没有其他的路径可以选择，只能进行全表扫描。最终的加锁情况，如下图所示：



如图，这是一个很恐怖的现象。首先，聚簇索引上的所有记录，都被加上了X锁。其次，聚簇索引每条记录间的间隙(GAP)，也同时被加上了GAP锁。这个示例表，只有6条记录，一共需要6个记录锁，7个GAP锁。试想，如果表上有1000万条记录呢？

在这种情况下，这个表上，除了不加锁的快照度，其他任何加锁的并发SQL，均不能执行，不能更新，不能删除，不能插入，全表被锁死。

当然，跟组合四：[id无索引, Read Committed]类似，这个情况下，MySQL也做了一些优化，就是所谓的semi-consistent read。semi-consistent read开启的情况下，对于不满足查询条件的记录，MySQL会提前放锁。针对上面的这个用例，就是除了记录[d,10]，[g,10]之外，所有的记录锁都会被释放，同时不加GAP锁。semi-consistent read如何触发：要么是read committed隔离级别；要么是Repeatable Read隔离级别，同时设置了innodb_locks_unsafe_for_binlog 参数。更详细的关于semi-consistent read的介绍，可参考我之前的一篇博客：MySQL+InnoDB semi-consitent read原理及实现分析 。

结论：在Repeatable Read隔离级别下，如果进行全表扫描的当前读，那么会锁上表中的所有记录，同时会锁上聚簇索引内的所有GAP，杜绝所有的并发 更新/删除/插入 操作。当然，也可以通过触发semi-consistent read，来缓解加锁开销与并发影响，但是semi-consistent read本身也会带来其他问题，不建议使用。

1. 组合九：Serializable

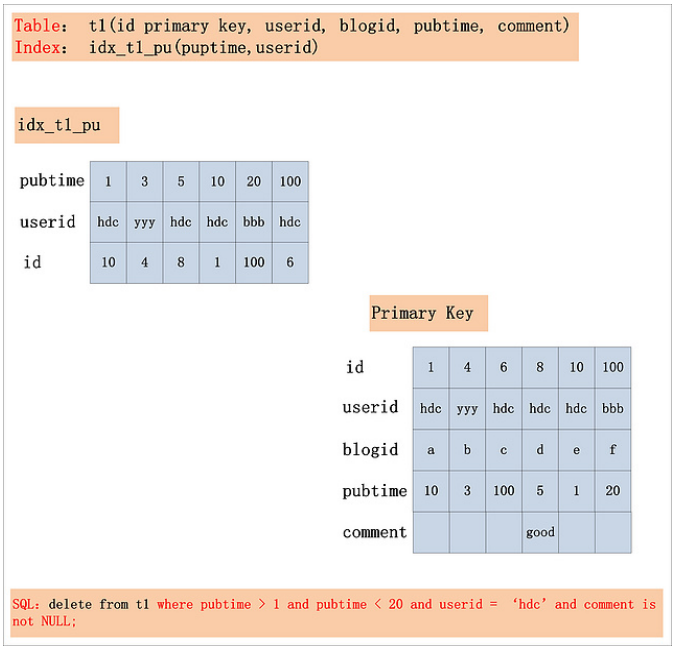
针对前面提到的简单的SQL，最后一个情况：Serializable隔离级别。对于SQL2：delete from t1 where id = 10; 来说，Serializable隔离级别与Repeatable Read隔离级别完全一致，因此不做介绍。

Serializable隔离级别，影响的是SQL1：select * from t1 where id = 10; 这条SQL，在RC，RR隔离级别下，都是快照读，不加锁。但是在Serializable隔离级别，SQL1会加读锁，也就是说快照读不复存在，MVCC并发控制降级为Lock-Based CC。

结论：在MySQL/InnoDB中，所谓的读不加锁，并不适用于所有的情况，而是隔离级别相关的。Serializable隔离级别，读不加锁就不再成立，所有的读操作，都是当前读。

1. 一条复杂的SQL

写到这里，其实MySQL的加锁实现也已经介绍的八八九九。只要将本文上面的分析思路，大部分的SQL，都能分析出其会加哪些锁。而这里，再来看一个稍微复杂点的SQL，用于说明MySQL加锁的另外一个逻辑。SQL用例如下：

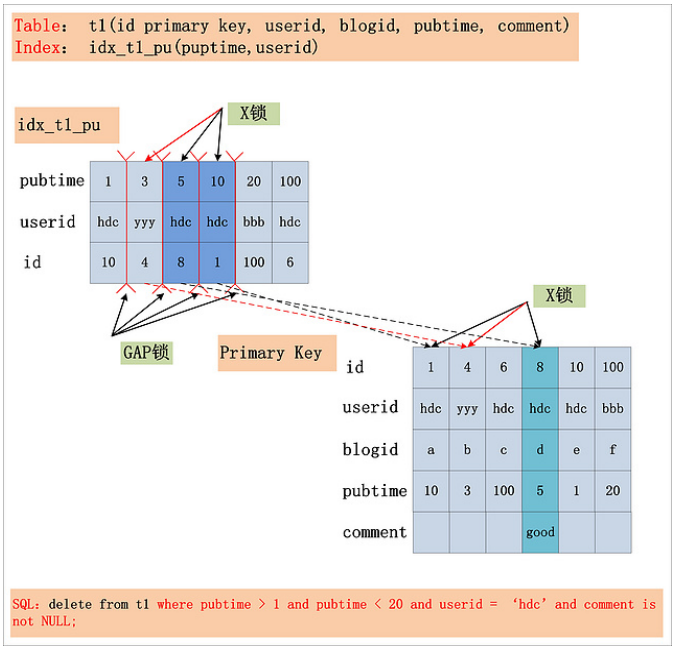


如图中的SQL，会加什么锁？假定在Repeatable Read隔离级别下 (Read Committed隔离级别下的加锁情况，留给读者分析。)，同时，假设SQL走的是idx_t1_pu索引。

在详细分析这条SQL的加锁情况前，还需要有一个知识储备，那就是一个SQL中的where条件如何拆分？具体的介绍，建议阅读我之前的一篇文章：SQL中的where条件，在数据库中提取与应用浅析 。在这里，我直接给出分析后的结果：

- **Index key :** pubtime > 1 and puptime < 20。此条件，用于确定SQL在idx_t1_pu索引上的查询范围。
- **Index Filter :** userid = 'hdc' 。此条件，可以在idx_t1_pu索引上进行过滤，但不属于Index Key。
- **Table Filter :** comment is not NULL。此条件，在idx_t1_pu索引上无法过滤，只能在聚簇索引上过滤。

在分析出SQL where条件的构成之后，再来看看这条SQL的加锁情况 (RR隔离级别)，如下图所示：



从图中可以看出，在Repeatable Read隔离级别下，由Index Key所确定的范围，被加上了GAP锁；Index Filter锁给定的条件 (userid = 'hdc')何时过滤，视MySQL的版本而定，在MySQL 5.6版本之前，不支持Index Condition Pushdown(ICP)，因此Index Filter在MySQL Server层过滤，在5.6后支持了Index Condition Pushdown，则在index上过滤。若不支持ICP，不满足Index Filter的记录，也需要加上记录X锁，若支持ICP，则不满足Index Filter的记录，无需加记录X锁 (图中，用红色箭头标出的X锁，是否要加，视是否支持ICP而定)；而Table Filter对应的过滤条件，则在聚簇索引中读取后，在MySQL Server层面过滤，因此聚簇索引上也需要X锁。最后，选取出了一条满足条件的记录[8,hdc,d,5,good]，但是加锁的数量，要远远大于满足条件的记录数量。

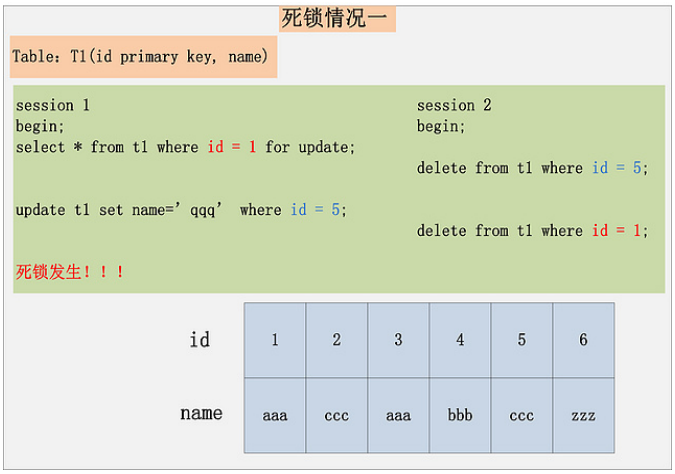
结论：在Repeatable Read隔离级别下，针对一个复杂的SQL，首先需要提取其where条件。Index Key确定的范围，需要加上GAP锁；Index Filter过滤条件，视MySQL版本是否支持ICP，若支持ICP，则不满足Index Filter的记录，不加X锁，否则需要X锁；Table Filter过滤条件，无论是否满足，都需要加X锁。

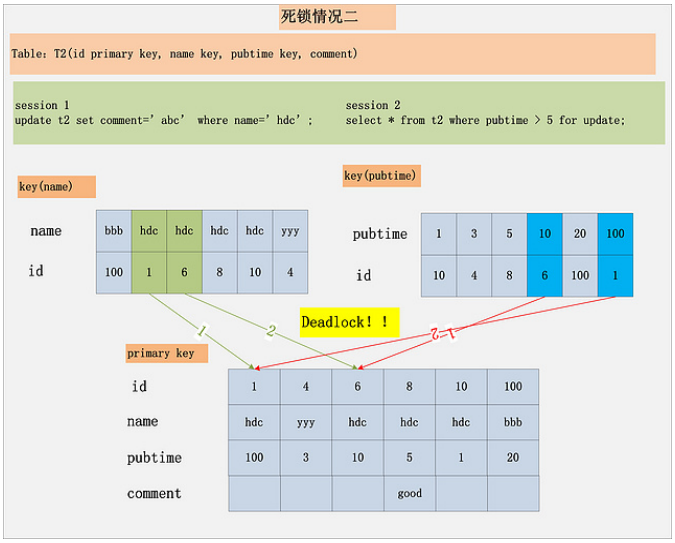
1. 死锁原理与分析

本文前面的部分，基本上已经涵盖了MySQL/InnoDB所有的加锁规则。深入理解MySQL如何加锁，有两个比较重要的作用：

- 可以根据MySQL的加锁规则，写出不会发生死锁的SQL；
- 可以根据MySQL的加锁规则，定位出线上产生死锁的原因；

下面，来看看两个死锁的例子 (一个是两个Session的两条SQL产生死锁；另一个是两个Session的一条SQL，产生死锁)：





上面的两个死锁用例。第一个非常好理解，也是最常见的死锁，每个事务执行两条SQL，分别持有了一把锁，然后加另一把锁，产生死锁。

第二个用例，虽然每个Session都只有一条语句，仍旧会产生死锁。要分析这个死锁，首先必须用到本文前面提到的MySQL加锁的规则。针对Session 1，从name索引出发，读到的[hdc, 1]，[hdc, 6]均满足条件，不仅会加name索引上的记录X锁，而且会加聚簇索引上的记录X锁，加锁顺序为先[1,hdc,100]，后[6,hdc,10]。而Session 2，从pubtime索引出发，[10,6],[100,1]均满足过滤条件，同样也会加聚簇索引上的记录X锁，加锁顺序为[6,hdc,10]，后[1,hdc,100]。发现没有，跟Session 1的加锁顺序正好相反，如果两个Session恰好都持有了第一把锁，请求加第二把锁，死锁就发生了。

结论：死锁的发生与否，并不在于事务中有多少条SQL语句，**死锁的关键在于**：两个(或以上)的Session**加锁的顺序**不一致。而使用本文上面提到的，分析MySQL每条SQL语句的加锁规则，分析出每条语句的加锁顺序，然后检查多个并发SQL间是否存在以相反的顺序加锁的情况，就可以分析出各种潜在的死锁情况，也可以分析出线上死锁发生的原因。

1. 总结

写到这儿，本文也告一段落，做一个简单的总结，要做的完全掌握MySQL/InnoDB的加锁规则，甚至是其他任何数据库的加锁规则，需要具备以下的一些知识点：

- 了解数据库的一些基本理论知识：数据的存储格式 (堆组织表 vs 聚簇索引表)；并发控制协议 (MVCC vs Lock-Based CC)；Two-Phase Locking；数据库的隔离级别定义 (Isolation Level)；
- 了解SQL本身的执行计划 (主键扫描 vs 唯一键扫描 vs 范围扫描 vs 全表扫描)；
- 了解数据库本身的一些实现细节 (过滤条件提取；Index Condition Pushdown；Semi-Consistent Read)；
- 了解死锁产生的原因及分析的方法 (加锁顺序不一致；分析每个SQL的加锁顺序)

有了这些知识点，再加上适当的实战经验，全面掌控MySQL/InnoDB的加锁规则，当不在话下。

标签: Database, Deadlock, Lock, MySQL
并发编程系列之一：锁的意义 C/C++ Volatile关键词深度剖析

1. 七月流火
   12月 13th, 201313:13
   回复 | 引用 | #1

   何大师，咨询个优化器的问题。select * from a,b group by a.name是先驱动表group by再join，还是先join再group by？我看几乎所有执行计划里都是前者，profile也看不出什么。何大师麻烦从源码角度解释下！多谢了！

   - catmonkeyxu
     12月 13th, 201315:21
     回复 | 引用 | #2

     是否因为group后再join，能够降低中间结果数量？

2. 

Rex
12月 13th, 201313:41
回复 | 引用 | #3

难得的数据库技术文章，对Mysql innodb事务与锁介绍的全面透彻。

3. 
ROKR
12月 13th, 201313:48
回复 | 引用 | #4

目前还用不上，但是介意我转载下么？以后学习可能有用

- 
hedengcheng
12月 13th, 201314:41
回复 | 引用 | #5

转载，请标明下出处吧。

  - 
hhkb4
6月 8th, 201619:51
回复 | 引用 | #6

讲的很通透，借用来给科普下，

4. 
dukope
12月 13th, 201314:50
回复 | 引用 | #7

深入潜出，nice!

5. 
gpfeng
12月 13th, 201323:23
回复 | 引用 | #8

可以针对insert专门出一篇，顺便介绍下隐式缩

- 
hedengcheng
12月 15th, 201317:08
回复 | 引用 | #9

这个建议，可以考虑。

  - 
and1
1月 6th, 201521:00
回复 | 引用 | #10

求大师 出一篇insert加锁的文章

6. 
lyxing
12月 14th, 201311:09
回复 | 引用 | #11

当前读，读取的是记录的最新版本，并且，当前读返回的记录，都会加上锁，保证其他事务不会再并发修改这条记录。

mysql正常关闭时，undo的脏块会写入datafile，undo会被释放吧？重启后，数据第一次被读取到buffer，此时应该是从datafile直接读取，应该为当前读吧？上锁？
可能我理解有误，请登成师解读下！

- 
hedengcheng
12月 15th, 201317:10
回复 | 引用 | #12

最新版本，必须保证是已经提交的一致版本，你说的情况，如果事物已提交，就读取。如果事务未提交，则通过undo回滚。

  - 
lyxing
12月 15th, 201321:13

回复 | 引用 | #13

假如已经提交，那么我觉得应该是当前读，这应该不上锁。

7. WenlongMeng
12月 14th, 201312:17
回复 | 引用 | #14

“在读多些少的OLTP应用中” => “在读多写少的OLTP应用中”

   - hedengcheng
12月 15th, 201317:11
回复 | 引用 | #15

谢谢指正。

8. fuyou001
12月 14th, 201316:15
回复 | 引用 | #16

请问聚簇索引上所有的记录，都被加上了X锁。无论记录是否满足条件，全部被加上X锁。这个锁的效果和表锁有什么区别？

   - hedengcheng
12月 15th, 201317:13
回复 | 引用 | #17

rc隔离级别下，有区别，记录仍旧可以插入。rr下，功能上无区别。但是innodb不会主动升级表锁。

     - ZoneJ
6月 16th, 201510:59
回复 | 引用 | #18

有个select的问题，select * from t1 where id=1 for update。这种情况对于组合1的加锁情况是怎样的？

       - ZoneJ
6月 16th, 201511:00
回复 | 引用 | #19

有个select的问题，select * from t1 where id>=1 and id <=10 for update。这种情况对于组合1的加锁情况是怎样的？

     - steven
1月 27th, 201609:27
回复 | 引用 | #20

在rc级别下,由于没有间隙锁,因此可以插入.
在rr级别下,由于有间隙锁,因此不能插入.
对吧?
但是什么时候,会触发表锁呢?

9. ken
12月 14th, 201317:23
回复 | 引用 | #21

非常感谢你的分享，受益匪浅：）我们也用MYSQL，只是简单的insert ，select，但数据量大，30T数据。只用myisam存的。

10. 东青
12月 14th, 201322:28
回复 | 引用 | #22

何大师留的思考题，测试过了，select for update 使用排他锁时，gap 锁是会生效的。

11. 东青
12月 15th, 201312:28
回复 | 引用 | #23

有一个关于死锁检测的问题，想请教一下 。
我在Mac Os 10.8版本下用了mysql Ver 14.14 Distrib 5.1.71, for apple-darwin12.0.0 (i386) 这个版本的mysql做了一个死锁检测实验。

表结构如下:
+——-+————-+——+——+———+——-+
| Field | Type | Null | Key | Default | Extra |
+——-+————-+——+——+———+——-+
| id | int(11) | NO | PRI | NULL | |
| name | varchar(10) | NO | | NULL | |
+——-+————-+——+——+———+——-+

我的事务隔离级别是repeatable-read.
现在表里面有两条记录：
+—-+——+
| id | name |
+—-+——+
| 1 | new |
| 4 | new |
+—-+——+

————————-实验1:————————————-
事务1：
begin;
select * from t where id =1 for update;

事务2：
begin;
update t set name =’ d’ where id =4 ;

事务1：
update t set name=’ d’ where id= 4;
(被Hold住)
事务2：
update t set name =’ d’ where id =1;

事务1 此时被检测到死锁，被重启事务了。
ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction

但是我做了另外一个测试：
————————实验2————————————
事务1：
begin
select * from t where id =1 for update
事务2：
begin
select * from t where id =4 for update;

事务1：
update t set name =’ d’ where id =4;
（被阻塞了 ）
事务2：
update t set name =’ d’ where id =1;
ERROR 1213 (40001): Deadlock …. （后面错误省略 ）

我想问的是，两个测试，mysql都检测到了死锁，为什么实验1中由事务2触发死锁，重启的是事务1；
但是实验2 中，事务2触发死锁，重启的却是事务2。 mysql在检测到死锁以后，重启的事务的依据是什么呢？
有什么好的死锁检测工具能推荐一下么？

- hedengcheng
  12月 15th, 201318:21
  回复 | 引用 | #24

  简单来说，mysql的死锁检测到之后，会选择一个事务进行回滚。而选择的依据：看哪个事务的权重最小，事务权重的计算方法：事务加的锁最少；事务写的日志最少；事务开启的时间最晚。实验1，事务2写了日志，事务1没有，回滚事务1。实验2，都没写日志，但是事务1开始的早，回滚事务2。

  死锁检测工具，目前我也不知道有什么好工具。

  - chinaxing
    11月 10th, 201514:22
    回复 | 引用 | #25

    事务被重启的操作，会报给应用程序吗？

12.

东青
12月 15th, 201321:04
回复 | 引用 | #26

谢谢回复，还会一如既往的关注你的技术文章的。

13. hongbin
12月 16th, 201312:15
回复 | 引用 | #27

看了你的分析，对MySQL加锁的知识又有了新的认识。非常感谢。
第二种死锁情况，为何不锁[20,100]?
最后总结处的知识点，能否给出一些参考链接或书籍，供深入学习，谢谢！

- hedengcheng
12月 16th, 201312:52
回复 | 引用 | #28

[20,100]也要加锁的，只是跟死锁无关，因此忽略了。

我考虑，添加一些参考资料，不过最好的资料，还是MySQL的源码。

14. 获思
12月 16th, 201314:01
回复 | 引用 | #29

非常好

15. jackwu
12月 18th, 201310:20
回复 | 引用 | #30

分析很详细，学习了。

16. benpaorun
12月 18th, 201311:12
回复 | 引用 | #31

最近刚好碰到了一个死锁问题，看了您这篇文章，收获颇多
但我碰到死锁的表是用的联合主键，想问下mysql的联合主键是如何建立索引的？
如果是建立两个非唯一索引，然后在MySQL Server层面验证主键唯一性的话，似乎无法解释我遇到的死锁问题
求教mysql联合主键的索引机制，谢谢啦！

- hedengcheng
12月 19th, 201309:22
回复 | 引用 | #32

联合主键，跟单一主键一模一样，没有区别。

  - benpaorun
12月 19th, 201317:53
回复 | 引用 | #33

不理解联合主键和单一主键一模一样啥意思，具体问题简化如下：
表结构为：
CREAT TABLE ‘t_gs_config’ {
‘serverId” int(11),
‘activityId’ int(11),
‘name’ varchar(255),
PRIMARY KEY(`serverId`, `activityId`)
} ENGINE=InnoDB DEFAULT CHARSET=utf8;
用的是联合主键serverId和activityId。
猜测当时场景，表t_gs_config的内容为
serverId activityId name
41 40 s11
42 40 s22
43 40 s13
75 45 s75
76 45 s76

77 45 s77
session1:
delete from t_gs_config where activityId=37;
session2:
insert into t_gs_config (name, activityId, serverId) values (‘s70’ , 44, 70);
insert into t_gs_config (name, activityId, serverId) values (‘s71’ , 44, 71);
insert into t_gs_config (name, activityId, serverId) values (‘s72’ , 44, 72);

我的理解是，在RR情况下
session1会在activityId的索引表上加一个gap锁，session2会在activityId和serverId的索引表上分别加一个gap锁，不会发生死锁啊
但实际情况是，死锁了。。。回滚了session1

- hedengcheng
  12月 20th, 201309:30
  回复 | 引用 | #34

  一模一样的意思，就是都是只有一个主键索引。只是一个主键只有一个列，一个有两个列。基本知识，建议先学习些数据库基础。

  - benpaorun
    12月 20th, 201311:39
    引用 | #35

    确实对数据库基础知识不熟悉，囧
    联合主键只有一个主键索引
    所以session1中的delete操作的约束条件只有一列，所以相当于组合8的情况，无索引，走的全表扫描，session2中的insert操作把包含了联合主键两列数据，所以相当于组合5的情况，依次加gap锁
    这样就能解释通了。。。谢谢啦！

    - hedengcheng
      12月 21st, 201318:26
      引用 | #36

      呵呵，这就是我这篇文章期待达到的效果。

- evergreen
  1月 27th, 201412:50
  回复 | 引用 | #37

  应该不会死锁吧，我做实验了，只会锁等待

  - hedengcheng
    1月 27th, 201414:00
    引用 | #38

    什么测试？针对什么场景？

  - evergreen
    1月 27th, 201417:47
    引用 | #39

    用的是和benpaorun一样的数据做的，执行sql的顺序也是一样的，发现只会锁等待。
    按照博文来的分析的话，我觉得锁等待也比较合理

17. wyh
    12月 18th, 201313:44
    回复 | 引用 | #40

    您好， 看了您的文章后，我立即分析了一下我遭遇到的死锁， 想请教您一下。

    Isolation-level: Read-Committed
    Table t(id primary key, status key, uuid)

    SQL:
    update t set uuid=” uuid” where status = 1 limit 50;

    这条sql 是多进程执行的， 也就是可能有两个 worker 在同时执行这条 SQL. 根据您的讲解， 因为status 不是 unique key ，所以会针对 status =1 的所有记录全部加锁, 并且 对应的 id 也会全部锁上； 如果有两个进程在同时操作，这就会导致 deadlock 吗？ 两个进程加锁的顺序应当是一致的啊， 也会出现死锁吗？ 还是有可能不一致？

另外，这里的 limit 50 应当和锁没有关系吧，还是说只找 50 条进行加锁呢？

- hedengcheng
  12月 19th, 201309:24
  回复 | 引用 | #41

  仅仅是这一条sql，加锁顺序是一致的，不会产生死锁。除非还有其他sql参与。

- hedengcheng
  12月 19th, 201309:24
  回复 | 引用 | #42

  还需要关注一下，这条sql走的是索引，还是全表扫描？

  - wyh
    12月 19th, 201321:42
    回复 | 引用 | #43

    应当是 索引吧， status 字段是有索引的，这样不会全表扫描吧。

    另外， 这里有 limit 50 是锁 50 条还是满足条件的都锁上呢？

    - hedengcheng
      12月 20th, 201309:31
      回复 | 引用 | #44

      有索引，没说就不会走全表扫描。有执行计划，有查询优化，有代价估算。

18. yesi
    12月 19th, 201317:50
    回复 | 引用 | #45

    delete from table1 where id in (1,2,3,4,5) ，id是主键，RR级别，这样会加GAP锁吗

    - hedengcheng
      12月 20th, 201309:32
      回复 | 引用 | #46

      你可以把这条sql，拆为5条id = 的sql来分析。

19. 郁白
    12月 23rd, 201314:22
    回复 | 引用 | #47

    在RC隔离级别下，索引表是否可以不加锁，而是在对数据表执行操作的时候执行double check，比如delete from t1 where id = 10; 在操作数据表的时候，检查索引列id是否还是10，如果不是就跳过

20. Seven
    12月 23rd, 201314:28
    回复 | 引用 | #48

    深入浅出，讲得特别好，继续关注博主的博客和微博

21. KingPoker
    12月 24th, 201322:26
    回复 | 引用 | #49

    组合七：id非唯一索引+RR
    比如一个Session A执行了delete from t1 where id = 10;会加两个锁，X锁和Gap锁
    Session B要执行什么样的sql？？才可以在Innodb_locks查看到Gap锁，还是说Gap锁在Innodb_locks就是表现为X锁？试过几种sql:
    delete from t1 where id =10;
    delete from t1 where name =’ b’ ;
    update t1 set id =111 where name =’ b’ ;

    请指教，谢谢！

22.

KingPoker
12月 24th, 201323:04
回复 | 引用 | #50

http://www.mysqlperformanceblog.com/2012/03/27/innodbs-gap-locks/
看完这个post知道了
Session B中执行insert into t1(name,id) values(‘ddd’,6);
可以查看到Gap锁

建议解释的时候能举个具体的sql例子，这样有图看文章还可以自己测试一把（对于不是很懂得人很有帮助），这样可以更好理解

- hedengcheng
  12月 25th, 201312:13
  回复 | 引用 | #51

  gap锁不是x锁，是两种不同的类型。关于你给的这个链接，组合七就很好的解释了这个问题。
  gap锁的功能，就是锁住此锁对应的区域，不可新插入数据。

  当然，你的建议不错，以后会注意新加一些例子。

  - KingPoker
    12月 25th, 201315:39
    回复 | 引用 | #52

    只是个人建议，你文章写得很好，思路非常清晰，看着就想动手试试，看看效果

    对应这个Gap锁有一点不是很明白，比如举得的例子，删除Id【等于】10，那为什么要锁住相邻的区域。
    你列举的数据，是不能插入id为6到11的数据，不在这个范围可以正常插入。
    在删除【等于】的情况下，Gap锁的区域范围是怎么限定了？是相邻的两个数（【6】，【11】）之间就不让插入？

    删除【大于】【小于】10锁定范围还是很好理解。

    - hedengcheng
      12月 26th, 201313:02
      回复 | 引用 | #53

      gap锁的目的，只要把握一点：就是让后续不能插入满足条件的新纪录，然后按照这个点，去考虑哪些地方需要加gap锁。

23. 李大玉
    12月 25th, 201310:59
    回复 | 引用 | #54

    能认真写这么多文字出来 还这么专业 不容易！！！！ 没有几个小时搞不定，学mysql的国人有福气啊

24. 七月流火
    12月 26th, 201316:40
    回复 | 引用 | #55

    关于组合七最后留的那个问题，（流火工程师）来解答下，此时会对不存在的间隙加上gap锁，比如表中已有id=1,2,3,6，那么select where id=4 for update会锁住(3,6)之间所有的间隙包括小数如3.1,3.2,5.999等，并且在show innodb status里面显示锁住的行为hex 80000006，只显示这一行信息。其他间隙如7,8,9不会锁的。

    - hedengcheng
      12月 26th, 201321:15
      回复 | 引用 | #56

      锁住间隙是对的，但是表述上有所不当。其实间隙是一个连续范围，例如你这里说的(3, 6)范围，就是一个连续范围，这个范围中的所有插入，均被禁止了。

25. 2013年个人微博推荐技术资料汇总 | 何登成的技术博客
    12月 26th, 201316:42
    #57

26. ontheway
    12月 26th, 201318:08
    回复 | 引用 | #58

    弱弱地问一句，我看的书里面都说的是RR隔离级别不允许脏读和不可重复读，但是可以幻读，怎么和作者说的不一样呢？

    -

hedengcheng
12月 26th, 201321:10
回复 | 引用 | #59

你说的没错，因此我在文章一开始，就强调了这一点。mysql innodb引擎的实现，跟标准有所不同。

- 笨笨爱琳琳
  1月 21st, 201419:48
  回复 | 引用 | #60

  这正是我这两天最大的困惑！
  一般大家提到Serializable才会来解决幻读的问题，RR只是解决不可重复读+脏读的问题。或者说，很多描述下，不可重复读有时候只是针对某条已经存在的记录，有时候是针对某个给定的条件。这么一看，确实明白了，Innodb的实现，与ISO定义的隔离级别及其解决问题的范围，是不一致的！

  - hedengcheng
    1月 22nd, 201409:37
    回复 | 引用 | #61

    嗯，是的。InnoDB在这个处理上，比较奇葩…

    - 笨笨爱琳琳
      1月 24th, 201413:26
      引用 | #62

      这问题应该算“奇葩”，还是算innodb做得更好？或者是对于“不可重复读”的理解和实现不一致？当然我不知道我理解的四层隔离级别及其对应处理的问题，是否和ANSI/ISO定义的是一致（希望回答一下）。我看其他博客中解释mysql（应该就是指InnoDB）的RR，解决了“幻读”问题。

      既然RR解决了“幻读”的问题，那么在mysql中，Serializable级别，应该解决哪些其他的问题呢？

      谢谢。

- Yhzhtk
  4月 14th, 201612:25
  回复 | 引用 | #63

  https://dev.mysql.com/doc/refman/5.6/en/set-transaction.html#isolevel_repeatable-read

  For locking reads (SELECT with FOR UPDATE or LOCK IN SHARE MODE), UPDATE, and DELETE statements, locking depends on whether the statement uses a unique index with a unique search condition, or a range-type search condition. For a unique index with a unique search condition, InnoDB locks only the index record found, not the gap before it. For other search conditions, InnoDB locks the index range scanned, using gap locks or next-key locks to block insertions by other sessions into the gaps covered by the range.

  官方的解释，RR级别时，如果查询是个范围，会加 gap locks 或者 next-key。 但是我不太确定是否会有例外，仍然不能避免幻读。

27. (转)MySQL 加锁处理分析 | 幽静的麦田_临沂php程序员
    12月 27th, 201314:27
    #64
28. joey
    1月 10th, 201413:06
    回复 | 引用 | #65

    您好，能不能介绍一下 insert into … select…from… 中 select 的 加锁方式？

    - hedengcheng
      1月 13th, 201409:53
      回复 | 引用 | #66

      insert into … select … from …中的select，会对表加读锁(更确切的说，应该是对表中所有的记录加读锁)，因此，select的操作，是一个当前读。

      - SeanSoong
        8月 13th, 201517:18
        回复 | 引用 | #67

        这个为什么要锁所有记录呢？不是可以MVCC实现多版本读吗？

29. ontheway
1月 13th, 201415:23
回复 | 引用 | #68

问题：如果组合五、组合六下，针对SQL：select * from t1 where id = 10 for update; 第一次查询，没有找到满足查询条件的记录，那么GAP锁是否还能够省略？
这个问题我在RR隔离级别下试过，感觉很奇怪，事务1执行select * from t1 where id = 10 for update; 后，事务2就不能插入insert id为10 的记录，说明是加了GAP锁，但是事务2也可以执行 select * from t1 where id = 10 for update;，这个时候事务1和事务2貌似都有id=10的GAP锁，2个事务都不能执行insert id=10的记录，会报死锁异常。给我的感觉就像是此时拿到的GAP锁不是一个排他锁，像是一个共享锁。作者能不能详细说明下这块。

- hedengcheng
1月 14th, 201409:28
回复 | 引用 | #69

gap锁本身的作用是防止后续的插入操作，因此gap锁只跟插入相冲突，gap锁之间不冲突，就会发生你这提到的情况。

  - ontheway
1月 26th, 201415:03
回复 | 引用 | #70

我下去针对GAP锁冲突做了如下测试，感觉GAP锁之间是有冲突，测试情况如下：
>>表article数据如下：
+——-+————-+
| id | name |
+——-+————-+
| 1 | title1 |
+——-+————-+
| 2 | title2 |
+——-+————-+
| 3 | title3 |
+——-+————-+
| 9 | title9 |
+——-+————-+
| 10 | title10 |
+——-+————-+

分别开2个mysql窗口模拟2个事务，设置autocommit=0，然后分别执行如下语句：
select * from article where [] for update;
+——————–+————–+————–+————–+——————–+
| | id=6 | id6 | id>5 and id<7 |
+——————–+————–+————–+————–+——————–+
| id=6 | 不冲突 | 不冲突 | 不冲突 | 不冲突 |
+——————–+————–+————–+————–+——————–+
| id6 | 不冲突 | 冲突 | 冲突 | 冲突 |
+——————–+————–+————–+————–+——————–+
| id>5 and id5 and id<7的时候也是加GAP锁，但是二个事务就会冲突，
这是什么原因？

    - ontheway
1月 26th, 201415:21
回复 | 引用 | #71

上面的内容被转码了，直接看图片吧，http://pan.baidu.com/s/1i34xAst

    - hedengcheng
1月 27th, 201411:03
回复 | 引用 | #72

你这个问题真的很好，也指出了我文中的一点小问题。

按照原理来说，id>5 and id<7这个查询条件，在表中找不到满足条件的项，因此会对第一个不满足条件的项(id = 9)上加GAP锁，防止后续其他事务插入满足条件的记录。

而GAP锁与GAP锁是不冲突的，那么为什么两个同时执行id>5 and id<7查询的事务会冲突呢？

原因在于，MySQL Server并没有将id<7这个查询条件下降到InnoDB引擎层，因此InnoDB看到的查询，是id>5，正向扫描。
读出的记录id=9，先加上next key锁(Lock X + GAP lock)，然后返回给MySQL Server进行判断。
MySQL Server此时才会判断返回的记录是否满足id<7的查询条件。此处不满足，查询结束。

因此，id=9记录上，真正持有的锁是next key锁，而next key锁之间是相互冲突的，这也说明了为什么两个id>5 and id<7查询的事务会冲突的原因。

- evergreen
  1月 27th, 201417:22
  引用 | #73

  id>6为何会与id<6冲突，一个间隙为(3, positive infinity)一个为(negative infinity, 9)；但是间隙锁不会冲突的啊

- hedengcheng
  1月 27th, 201418:31
  引用 | #74

  看我给ontheway的回复。查询过程中，不是只加间隙锁，而是加next key，next key是会冲突的。

30. 周波
    1月 20th, 201415:35
    回复 | 引用 | #75

    关于MySQL的锁有些地方没搞清楚
    1. 意向锁的作用
    SELECT … FOR UPDATE 会锁住选中的行，和直接对那些行加X锁有什么区别？
    2. S和X锁什么时候加？
    在RR级别下的SELECT好像不会加S锁，但是会加X锁。那什么时候会加S锁呢？

    - 周波
      1月 20th, 201415:59
      回复 | 引用 | #76

      补充一下，在网上找到一些资料说Intention lock 是表级锁，但是我这边试下来，在RR级别下是next key lock，并不会锁住整张表。

      - hedengcheng
        1月 22nd, 201409:38
        回复 | 引用 | #77

        谁说Intention Lock是表级锁？谁写的？拉出去突突了。

        - now
          9月 30th, 201415:14
          回复 | 引用 | #78

          Intention Lock是用在哪个级别上的？

    - hedengcheng
      1月 22nd, 201409:39
      回复 | 引用 | #79

      看你这些问题，我真不知道如何回答，感觉你对锁的理解，比较混乱。至于第二点，select lock in share mode; 会加S锁。

      - 周波
        1月 22nd, 201414:57
        回复 | 引用 | #80

        我理解的select lock in share mode会加IS锁，select for update会加IX锁。
        所以我没搞清楚IX锁和X锁，他们的区别在哪里。都是对一些记录加锁。

        - hedengcheng
          1月 23rd, 201409:48
          回复 | 引用 | #81

          建议去看一些锁相关的基础文章，你这些理解，都是有问题的。

31. 笨笨爱琳琳
    1月 22nd, 201410:08
    回复 | 引用 | #82

登博，帮你搞了一个净版的pdf，见：http://yun.baidu.com/share/link?shareid=3229086908&uk=4146200807#dir/path=%2F%E5%85%A8%E5%B1%80%E5%85%B1%E4%BA%AB

- hedengcheng
  1月 23rd, 201409:47
  回复 | 引用 | #83

  🙂

32. 润洁
    1月 22nd, 201421:37
    回复 | 引用 | #84

    理论上不应该出现死锁的死锁，请帮忙诊断一下吧，多谢！
    场景：
    两个或多个事务同时删除同一条记录，使用的条件是一个唯一索引的值，但不是主键。
    表结构：
    CREATE TABLE dltask (
    id bigint unsigned NOT NULL AUTO_INCREMENT COMMENT ‘auto id’,
    a varchar(30) NOT NULL COMMENT ‘uniq.a’,
    b varchar(30) NOT NULL COMMENT ‘uniq.b’,
    c varchar(30) NOT NULL COMMENT ‘uniq.c’,
    x varchar(30) NOT NULL COMMENT ‘data’,
    PRIMARY KEY (id),
    UNIQUE KEY uniq_a_b_c (a, b, c)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT=’deadlock test’;
    事务隔离级别：
    REPEATABLE READ
    事务内仅有的一条SQL语句：
    delete from dltask where a=? and b=? and c=?;

    5.5.20 版本的死锁日志：
    ————————
    LATEST DETECTED DEADLOCK
    ————————
    140122 18:11:58
    *** (1) TRANSACTION:
    TRANSACTION 930F9, ACTIVE 0 sec starting index read
    mysql tables in use 1, locked 1
    LOCK WAIT 2 lock struct(s), heap size 376, 1 row lock(s)
    MySQL thread id 2096, OS thread handle 0x7f3570976700, query id 1485879 localhost 127.0.0.1 rj updating
    delete from dltask where a=’b’ and b=’b’ and c=’a’
    *** (1) WAITING FOR THIS LOCK TO BE GRANTED:
    RECORD LOCKS space id 0 page no 12713 n bits 96 index `uniq_a_b_c` of table `dltst`.`dltask` trx id 930F9 lock_mode X waiting
    *** (2) TRANSACTION:
    TRANSACTION 930F3, ACTIVE 0 sec starting index read
    mysql tables in use 1, locked 1
    3 lock struct(s), heap size 376, 2 row lock(s)
    MySQL thread id 2101, OS thread handle 0x7f3573d88700, query id 1485872 localhost 127.0.0.1 rj updating
    delete from dltask where a=’b’ and b=’b’ and c=’a’
    *** (2) HOLDS THE LOCK(S):
    RECORD LOCKS space id 0 page no 12713 n bits 96 index `uniq_a_b_c` of table `dltst`.`dltask` trx id 930F3 lock_mode X locks rec but not gap
    *** (2) WAITING FOR THIS LOCK TO BE GRANTED:
    RECORD LOCKS space id 0 page no 12713 n bits 96 index `uniq_a_b_c` of table `dltst`.`dltask` trx id 930F3 lock_mode X waiting
    *** WE ROLL BACK TRANSACTION (1)

    5.6.15 版本的死锁日志：
    ————————
    LATEST DETECTED DEADLOCK
    ————————
    2014-01-22 20:48:08 7f4248516700
    *** (1) TRANSACTION:
    TRANSACTION 2268, ACTIVE 0 sec starting index read
    mysql tables in use 1, locked 1
    LOCK WAIT 2 lock struct(s), heap size 376, 1 row lock(s)
    MySQL thread id 11, OS thread handle 0x7f4248494700, query id 1207 localhost 127.0.0.1 rj updating
    *** (1) WAITING FOR THIS LOCK TO BE GRANTED:
    RECORD LOCKS space id 6 page no 4 n bits 96 index `uniq_a_b_c` of table `dltst`.`dltask` trx id 2268 lock_mode X locks rec but not gap waiting
    *** (2) TRANSACTION:
    TRANSACTION 2271, ACTIVE 0 sec starting index read
    mysql tables in use 1, locked 1

3 lock struct(s), heap size 376, 2 row lock(s)
MySQL thread id 9, OS thread handle 0x7f4248516700, query id 1208 localhost 127.0.0.1 rj updating
delete from dltask where a=’b’ and b=’a’ and c=’c’
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 6 page no 4 n bits 96 index `uniq_a_b_c` of table `dltst`.`dltask` trx id 2271 lock_mode X locks rec but not gap
*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 6 page no 4 n bits 96 index `uniq_a_b_c` of table `dltst`.`dltask` trx id 2271 lock_mode X waiting
*** WE ROLL BACK TRANSACTION (1)

- 
  hedengcheng
  1月 23rd, 201411:21
  回复 | 引用 | #85

  几个问题：
  1. 此问题重现频率如何，高不？

  2. 确定每个事务都只有一条delete语句？

  3. 确定delete走的是uniq_a_b_c索引？

  4. 针对delete from dltask where a=’b’ and b=’b’ and c=’a’ 删除操作，表中有没有找到对应的记录？

  - 
    润洁
    1月 23rd, 201412:52
    回复 | 引用 | #86

    1. 线上主要的机率不高，毕竟不同请求会操作同一条记录的机率很小，操作同一条的情况出现死锁的就更小一些，因为总有个时机问题嘛。但确实偶尔就出现。
    这里发出来的，是针对线上问题做的一个完全等价的模拟，模拟程序提高了多个线程删除同一条记录的机率，所以死锁就特别多。
    2. 确定只有这一条delete语句。
    3. 用 explain select * from dltask where a=? and b=? and c=? 看，是走了这个唯一索引的。
    4. 线上的，相应的记录在表中不一定存在，但等价的模拟程序里，相应的记录在删除之前，在表中肯定是存在的。

    - 
      润洁
      1月 23rd, 201412:53
      回复 | 引用 | #87

      不好意思，错字了：
      1. 线上主要的机率不高 -> 线上出现的机率不高

      - 
        hedengcheng
        1月 23rd, 201416:11
        引用 | #88

        是AutoCommit吗？还是有begin commit？ 有测试脚本不？

      - 
        hedengcheng
        1月 23rd, 201418:31
        引用 | #89

        我对这个问题很感兴趣，有其他联系方式吗？讨论下。

33. 
    润洁
    1月 23rd, 201419:07
    回复 | 引用 | #90

    AutoCommit 无影响，true 或者 false 效果都一样。
    是Java程序，模拟程序也同样是用Java写的，如果您觉得有必要使用这个测试程序，就往我邮箱里发个消息吧，我用附件的形式把测试的程序发给您。
    我很佩服您对问题的分析，透彻清晰有条理，所以才把这个问题发上来的，如果觉得有必要，我可以在回邮件的时候带上电话。

    - 
      hedengcheng
      1月 23rd, 201419:10
      回复 | 引用 | #91

      这个问题有点颠覆了我对InnoDB锁实现的认识，因此一定要搞清楚，我给你发邮件，可以的话，把测试程序给我看一下吧。

- 润洁
  1月 23rd, 201422:25
  回复 | 引用 | #92

  邮件已发，谢谢！

- hedengcheng
  1月 23rd, 201420:08
  回复 | 引用 | #93

  我大概知道原因了，明天再确认下。

  - 润洁
    1月 23rd, 201422:27
    回复 | 引用 | #94

    静侯佳音

    - hedengcheng
      1月 27th, 201411:04
      回复 | 引用 | #95

      可见我最新博文的分析：《一个最不可思议的MySQL死锁分析》 http://hedengcheng.com/?p=844

34. 一个最不可思议的MySQL死锁分析 | 何登成的技术博客
    1月 24th, 201415:54
    #96
35. MySQL 加锁处理分析 | Steve Guan
    1月 26th, 201420:21
    #97
36. taowei
    2月 4th, 201421:06
    回复 | 引用 | #98

    何大师，你写的很好啊，能否再细致些啊，比如在8种情况下 分别对二级索引和主键索引加了何种锁，比如第三种情况，对id 、 name 分别加的是哪一种锁？

37. peiyuc
    2月 17th, 201421:56
    回复 | 引用 | #99

    在RR+非唯一索引的情况下，在两个事务中可以同时执行有共同gap锁区间的sql，是怎么回事呢？就是您所说的组合七中的例子，insert [10,aa] 是可以执行的

    - hedengcheng
      2月 18th, 201410:46
      回复 | 引用 | #100

      Gap lock之间是不相互冲突的。能举个例子吗？

38. +01
    2月 23rd, 201416:54
    回复 | 引用 | #101

    大师，我有两个观点和您不同，请指正
    1.id非唯一索引，RR下，对所有满足条件的记录加next-key lock，这样可以保证每个记录之前不会插入幻读行，然后再最后一个满足条件的记录之后再加一个gap lock，防止尾部添加幻读行
    2.无索引的时候，我的理解是加表锁。因为任何写操作都要先申请写意向锁，意向锁是表锁，IX与表X是互斥的，这样可以做到表在加锁的时候不会被写入。虽然我没读过源码，但觉得如果把所有记录都加锁没有这个必要，意向锁的设计就是为了解决行锁和表锁冲突的。

    - hedengcheng
      2月 24th, 201409:21
      回复 | 引用 | #102

      “意向锁的设计就是为了解决行锁和表锁冲突的” ，原则性错误，建议再看一些锁相关的资料。

+01
2月 24th, 201412:33
回复 | 引用 | #103

这个只是我的个人理解，因为意向锁本身是表锁且意向锁之间没有冲突的，意向锁和行锁存在不兼容。如果意向锁不是在表被锁的时候阻止行锁的请求，那意向锁设计的意义是为了什么呢？。无索引的时候加表锁，这个理解是否有问题？如果是对所有记录加锁，那innodb是否在dml语句里就不会对表加锁呢？

hedengcheng
2月 25th, 201409:38
回复 | 引用 | #104

“那innodb是否在dml语句里就不会对表加锁呢？”这个理解是对的，InnoDB在dml语句时，只会加表意向锁，不会加真正的表锁。

+01
2月 25th, 201416:04
引用 | #105

大师，我扔有疑问，据我了解，死锁有另一个原因：申请锁的数量太多。如果无索引的时候对每个记录都加锁，那是否会因为申请锁太多而导致死锁呢？我曾经在无索引的情况下更新过亿级别的表，没有造成死锁，我使用的mysql版本是5.1.63，innodb不是plugin的，和版本有关系吗？

hedengcheng
2月 25th, 201418:46
引用 | #106

死锁的原因，就是两线程以上，加锁的顺序相反，跟锁数量多少没有任何关系。

39. randy.su
4月 17th, 201422:31
回复 | 引用 | #107

大师好啊，我现在的情况是这样：多个进程对一张表t进行并发的批量更新操作，比如：update t set t.status=’0’ where t.id in (${ids})，表t没有主键，id有非唯一索引，其中${id}是逗号分隔的id串，比如：
进程1：update t set t.status=’0’ where t.id in (1,3,5,7,9})
进程2：update t set t.status=’0’ where t.id in (2,4,6,8,10})
进程3：update t set t.status=’0’ where t.id in (11,12,13,14,15})
进程4：....
我们保证了各自进程查询出的记录是不会跟别的进程重复的，这样会发生死锁，为什么呢？

40. randy.su
4月 17th, 201422:37
回复 | 引用 | #108

> **randy.su :**
> 大师好啊，我现在的情况是这样：多个进程对一张表t进行并发的批量更新操作，比如：update t set t.status=’0’ where t.id in (${ids})，表t没有主键，id有非唯一索引，其中${id}是逗号分隔的id串，比如：
> 进程1：update t set t.status=’0’ where t.id in (1,3,5,7,9})
> 进程2：update t set t.status=’0’ where t.id in (2,4,6,8,10})
> 进程3：update t set t.status=’0’ where t.id in (11,12,13,14,15})
> 进程4：....
> 我们保证了各自进程查询出的记录是不会跟别的进程重复的，这样会发生死锁，为什么呢？

进程1：update t set t.status=’0’ where t.id in (1,3,5,7,9)
进程2：update t set t.status=’0’ where t.id in (2,4,6,8,10)
进程3：update t set t.status=’0’ where t.id in (11,12,13,14,15)
进程4：....

41. randy.su
4月 18th, 201416:59
回复 | 引用 | #109

> **randy.su :**
>
> > **randy.su :**
> > 大师好啊，我现在的情况是这样：多个进程对一张表t进行并发的批量更新操作，比如：update t set t.status=’0’ where t.id in (${ids})，表t没有主键，id有非唯一索引，其中${id}是逗号分隔的id串，比如：

进程1：update t set t.status=’0’ where t.id in (1,3,5,7,9})
进程2：update t set t.status=’0’ where t.id in (2,4,6,8,10})
进程3：update t set t.status=’0’ where t.id in (11,12,13,14,15})
进程4：….
我们保证了各自进程查询出的记录是不会跟别的进程重复的，这样会发生死锁，为什么呢？

进程1：update t set t.status=’0’ where t.id in (1,3,5,7,9)
进程2：update t set t.status=’0’ where t.id in (2,4,6,8,10)
进程3：update t set t.status=’0’ where t.id in (11,12,13,14,15)
进程4：….

上面讲错了，有主键，是另外一个字段，比如key，而id是具有非唯一性索引的字段。
t表结构：

```
CREATE TABLE `t` (
`key` bigint(20) NOT NULL AUTO_INCREMENT,
`id` bigint(20) DEFAULT NULL,
`name` varchar(60) DEFAULT NULL,
`createTime` datetime DEFAULT NULL,
PRIMARY KEY (`key`),
KEY `idx_id` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=50000 DEFAULT CHARSET=utf8;
```

- hedengcheng
  4月 28th, 201409:09
  回复 | 引用 | #110

  这个无法保证不产生死锁，无论是rr还是rc隔离级别。

42. Atlas
    5月 20th, 201415:37
    回复 | 引用 | #111

    真心的好啊，深入浅出，纠正了我很多理解错误的地方。继续关注。继续膜拜

43. java文盲
    7月 9th, 201418:45
    回复 | 引用 | #112

    RECORD LOCKS space id 2523 page no 4 n bits 72 index `uid_itemType_itemId_uinque` of table `payment`.`test_dead_lock` trx id A41BEFA lock_mode X insert intention waiting

    请问插入意向锁是不是不区分rr和rc，我之前理解的是插入意向锁属于gap锁 ，如果用rc不就相当于关掉gap锁了吗？那为什么rc级别下还会出现insert intention waiting。。。

    - 大自在真人
      7月 10th, 201422:42
      回复 | 引用 | #113

      遇到了相同的问题，死锁冲突，二级唯一索引上，单条inser语句事务。一个事务锁定 lock_mode X locks rec but not gap，然后请求 lock_mode X locks gap before rec insert intention waiting，另一个insert语句 lock mode S waiting，所有的锁请求都是同一数据文件同一数据块的同一位置，然后第二个事务就因为死锁被杀掉了，很是奇怪，麻烦解说一下，谢谢。

      - 大自在真人
        7月 11th, 201406:05
        回复 | 引用 | #114

        表上有三个索引，主键索引，一个唯一索引，一个非唯一索引，但报告锁冲突的是唯一索引

44. 桦仔
    7月 23rd, 201417:23
    回复 | 引用 | #115

    Intention lock 不一定是表级锁
    我个人理解是更大范围的锁，但不一定是表锁
    我从sqlserver转过来的
    http://www.cnblogs.com/shanksgao/p/3140149.html?ADUIN=1815357042&ADSESSION=1405653123&ADTAG=CLIENT.QQ.5335_.0&ADPUBNO=26366
    比如表做了分区，那么就是分区锁，虽然作用的对象在表上
    我从sqlserver转过来的

45. leezq
8月 11th, 201409:30
回复 | 引用 | #116

HI,博主，您好。有个地方没看懂，能否解释下。组合四：从图中可以看到，满足删除条件的记录有两条，但是，聚簇索引上所有的记录，都被加上了X锁。无论记录是否满足条件，全部被加上X锁。既不是加表锁，也不是在满足条件的记录上加行锁。 既然锁定了所有的聚簇索引的全部记录，这不就是表锁吗？

- hedengcheng
4月 13th, 201521:05
回复 | 引用 | #117

表锁就是真正的表锁，哪怕我把表中所有记录都加上了行锁，也不能说就是加上表锁了。

46. Beginning
9月 12th, 201418:16
回复 | 引用 | #118

登博，你好，有个问题想请教下，关于组合七中，会加gap lock，比如那里[6,c]和[10,b]之间加了gap锁，也就是说能够插入[6,a] 但是不能插入[6,d]，为何gap锁要这样加呢？在哪里能够查到gap锁范围这方面的资料？希望网易前同事能够指点下，嘿嘿

- hedengcheng
4月 13th, 201521:06
回复 | 引用 | #119

gap锁范围，简单来说，就是锁当前记录和上一条记录之间的空白区域。

47. 简
11月 12th, 201409:36
回复 | 引用 | #120

何博，我有这样一个问题，希望您能帮解答一下，问题如下：
有这样一个存储过程，
BEGIN
DECLARE myid INT DEFAULT 0;
DECLARE myname CHAR(50) DEFAULT ” ;

START TRANSACTION;

SELECT * FROM test ORDER BY id LIMIT 1 FOR UPDATE;

SELECT id,d INTO myid, myname FROM test2 WHERE c = 0.4 ORDER BY id LIMIT 1;

UPDATE test2 SET c = 09 WHERE id = myid;

INSERT test3(name) VALUES(myname);

COMMIT;

END
业务前提，test2中的d字段数据不重复，而test3中不能有test2中d字段的重复数据，如果在高并发下，发现test3有了test2字段d的重复数据，说明test2被两个session执行过，如果我在“SELECT id,d INTO myid, myname FROM test2 WHERE c = 0.4 ORDER BY id LIMIT 1;” 这句中加上for update确没有问题。在我的理解是，test1中使用了锁应该是处于竞争锁而等待，直到一个sp运行完才会释放锁，为什么会test2中的并发，麻烦您帮解答下

- 简
11月 12th, 201409:39
回复 | 引用 | #121

问题补充，隔离等级是RR，test1，test2中的id都是PK，两个表都没有使用二级索引

48. xep
11月 13th, 201412:22
回复 | 引用 | #122

组合五：id主键+RR
应该也加了gap lock
测试： s1:
(root@localhost) [db_t1]> select * from test1;
+—-+——-+———————+

```
| id | name | cdate |
+—-+——-+———————+
| 1 | test1 | 2014-11-12 14:32:41 |
| 2 | test2 | 2014-11-13 11:43:01 |
| 3 | test3 | 2014-11-13 10:58:12 |
+—-+——-+———————+
3 rows in set (0.00 sec)

(root@localhost) [db_t1]> update test1 set name =  'test22'  where id=2;
(root@localhost) [db_t1]> commit;

s2:
(root@localhost) [db_t1]> select * from test1;
+—-+——-+———————+
| id | name | cdate |
+—-+——-+———————+
| 1 | test1 | 2014-11-12 14:32:41 |
| 2 | test2 | 2014-11-13 11:43:01 |
| 3 | test3 | 2014-11-13 10:58:12 |
+—-+——-+———————+
3 rows in set (0.00 sec)

(root@localhost) [db_t1]> commit;
Query OK, 0 rows affected (0.00 sec)

(root@localhost) [db_t1]> select * from test1;
+—-+——-+———————+
| id | name | cdate |
+—-+——-+———————+
| 1 | test1 | 2014-11-12 14:32:41 |
| 2 | test22 | 2014-11-13 11:43:37 |
| 3 | test3 | 2014-11-13 10:58:12 |
+—-+——-+———————+
```

49. 
CC
12月 1st, 201416:44
回复 | 引用 | #123

请教登博，博客中说的那些是否有index的判断是在storage层还是在sql层呢，因为如果delete的话在storage的接口应该是delete_row()。我现在正在看storage中的heap，因为才开始，想找个比较简单的看看先熟悉熟悉，希望能从你那得到点建议。

50. 
nick
12月 5th, 201413:38
回复 | 引用 | #124

对于案例8，RR级别下无索引锁住所有记录和gap，为何不直接加table lock呢？innodb的table lock和myisam的实现方式是否一样？在innodb中直接用命令lock table xx read会调用row lock吗？

51. 
shun
12月 23rd, 201417:26
回复 | 引用 | #125

期待出书《MySQL源码剖析》，马上预订！

52. 
prclqz
1月 30th, 201511:57
回复 | 引用 | #126

学长，关于MySQL的意向锁有点搞不懂，能有机会解释下吗？

53. 
trikker
2月 5th, 201514:53
回复 | 引用 | #127

文章讲得非常好，深入浅出。只是做了实验之后，结果和你的“一条复杂的SQL加锁分析”有一点不一样。
前提都和你的一样，事务隔离级别是RR，innodb_locks_unsafe_for_binlog是关闭的。
官方的mysql-5.6.21
实验结果如下：

首先数据是和你一样的。
mysql> show create table t1\G

*************************** 1. row ***************************
Table: t1
Create Table: CREATE TABLE `t1` (
`id` int(11) NOT NULL,
`userid` varchar(20) DEFAULT NULL,
`blogid` varchar(20) DEFAULT NULL,
`pubtime` int(11) DEFAULT NULL,
`comment` varchar(20) DEFAULT NULL,
PRIMARY KEY (`id`),
KEY `idx_t1_pu` (`pubtime`,`userid`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.04 sec)

mysql> select * from t1;
+——+———+———+———+———+
| id | userid | blogid | pubtime | comment |
+——+———+———+———+———+
1	hdc	a	10	NULL
4	yyy	b	3	NULL
6	hdc	c	100	NULL
8	hdc	d	5	good
10	hdc	e	1	NULL
100	bbb	f	20	NULL
+——+———+———+———+———+
6 rows in set (0.00 sec)

然后，那条delete语句的执行计划如下，可以看出没有用到ICP：
mysql> desc delete from t1 where pubtime>1 and pubtime begin;
Query OK, 0 rows affected (0.00 sec)

mysql> delete from t1 where pubtime>1 and pubtime select * from t1 where id=100 for update;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction

从这里可以看出，MySQL没有使用ICP，所以按照你的分析是锁了3行索引记录（对应到主键上面也是锁3行记录），但是实验表明id=100的那行记录也被锁了。

也就是说,delete from t1 where pubtime<20;(假设pubtime上面有非唯一索引且存在pubtime=20这条记录)，这样的where条件pubtime<20是会锁pubtime=20这条记录的，这和你的文章说的不一样。

我的推理也和你一样，不应该锁这条记录，但是MySQL就是锁了。为什么？？？？

- trikker
  2月 5th, 201514:57
  回复 | 引用 | #128

  我的评论不知道为什么很多行的字消失了，所以看上去好像没有逻辑，不知道你的评论系统出了什么状况。反正就是，delete from t1 where pubtime<20;(假设pubtime上面有非唯一索引且存在pubtime=20这条记录)，这样的where条件pubtime<20是会锁pubtime=20这条记录的，这和你的文章说的不一样。我的推理也和你一样，不应该锁这条记录，但是MySQL就是锁了。为什么？？？？我可以发邮件给你的，但是我现在不知道你的邮箱！

  - trikker
    2月 5th, 201515:07
    回复 | 引用 | #129

    更具体点说，就是实际上那条复杂的delete语句delete from t1 where pubtime >1 and pubtime <20 and userid='hdc' and comment is not null;锁住的并非只是pubtime=3、5、10这3条记录（没有使用ICP情况下），还有pubtime=20这条记录，但是没有锁pubtime=1这条记录。不知道原因！

54. Huyet
    2月 13th, 201511:50
    回复 | 引用 | #130

    对于组合四
    begin;
    delete from t where id=10;
    不commit

    然后查看状态，会看到锁住了7行，而不是2行？
    就是说，文中提到的优化没效果。
    我理解的优化就是应该只锁住2行，其他不满足条件的都防锁了.

    - Huyet

2月 13th, 201514:25
回复 | 引用 | #131

唉，原来是自己搞错，mysql默认是rr，我就动态修改为rc后，出现上面那种问题。后来我直接改了配置文件再时，在rc下，组合四中，的确有优化，最后只锁了符合条件的行。是我动态修改系统变量的写法出问题。。。

- 
  Richard
  6月 11th, 201510:59
  回复 | 引用 | #132

  RR主要是防止幻象才加的gap锁。不过官方不建议改默认级别。 不过能把控的话，什么级别都无所谓

55. 
MYSQL菜鸟
3月 2nd, 201516:22
回复 | 引用 | #133

你好，你的文章我看了好几遍了，但是关于RR级别组合七那种，按照您说的应该是加了那几条记录的行级锁，比如id为主键 name 为非唯一索引,先执行 delete from person where name = ‘zhang’，然后这个事务不提交，再在另外一个事务里执行 delete from person where name = ‘li’ 就会锁等待呢，我如果执行delete from person where id = ‘2’就可以成功。 假定有2条数据 id = ‘1’ name = ‘zhang’ 和 id=’ 2’ name=’ li’ . 希望您能抽空解答 谢了 我qq 785848201

- 
  hedengcheng
  4月 13th, 201521:00
  回复 | 引用 | #134

  你再仔细想想，这个问题我其实写的比较明白了

  - 
    mysqlzac
    5月 13th, 201516:20
    回复 | 引用 | #135

    这个是因为gap只会出现在二级index中

56. 
alexniver
3月 12th, 201515:29
回复 | 引用 | #136

牛逼…… 之前遇到死锁都晕得不行~~ 这回有解决的思路了

57. 
Bin
3月 26th, 201520:48
回复 | 引用 | #137

读了博主的文章，解惑了，非常感谢！

58. 
i9944
4月 4th, 201511:41
回复 | 引用 | #138

有个死锁log麻烦大家看下 我不明白事务2 为什么获取了一个锁后 为什么还会获取这个锁 导致等待呢

LATEST DETECTED DEADLOCK
————————
150319 11:55:31
*** (1) TRANSACTION:
TRANSACTION 37BA66104, ACTIVE 0 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 5 lock struct(s), heap size 1248, 4 row lock(s), undo log entries 1
MySQL thread id 386472597, OS thread handle 0x7f893b061700, query id 11094033559 10.64.29.226 hotel_service statistics
select * from `tab`
WHERE id = 3863754
for update
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 250 page no 46518 n bits 280 index `PRIMARY` of table `tab`.`tab` trx id 37BA66104 lock_mode X locks rec but not gap waiting
*** (2) TRANSACTION:
TRANSACTION 37BA66106, ACTIVE 0 sec starting index read
mysql tables in use 1, locked 1
5 lock struct(s), heap size 1248, 4 row lock(s), undo log entries 1

MySQL thread id 386481878, OS thread handle 0x7f8c0a0c2700, query id 11094033564 10.64.17.221 hotel_service statistics
select * from `tab`
WHERE id = 3863753
for update
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 250 page no 46518 n bits 280 index `PRIMARY` of table `tab`.`tab` trx id 37BA66106 lock_mode X locks rec but not gap
*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 250 page no 46518 n bits 280 index `PRIMARY` of table `tab`.`tab` trx id 37BA66106 lock_mode X locks rec but not gap waiting
*** WE ROLL BACK TRANSACTION (2)
————
TRANSACTIONS
————
Trx id counter 37BA87F6F
Purge done for trx' s n:o = 37BA87ED2, sees < 37BA87ED2
————————————-
END OF INNODB MONITOR OUTPUT
============================

59. 
xiangzi
4月 14th, 201515:56
回复 | 引用 | #139

id非唯一索引+RC

create table tidx (id int, name varchar(100));
create index idx_tidx on tidx(id);
insert into tidx values (10, 'zhao');
insert into tidx values (11, 'qian');
insert into tidx values (12, 'sun');

begin;
delete from tidx where id=10;

查看事务
trx_rows_locked: 3

与描述的不是期待的4行

使用相同表结构和数据，trx_rows_locked是5个，而不是期待的4个。

或者是我的期待有问题？

mysql的版本是5.6.21

- 
xiangzi
4月 14th, 201516:09
回复 | 引用 | #140

> **xiangzi :**
> id非唯一索引+RC
> create table tidx (id int, name varchar(100));
> create index idx_tidx on tidx(id);
> insert into tidx values (10, 'zhao');
> insert into tidx values (11, 'qian');
> insert into tidx values (12, 'sun');
> begin;
> delete from tidx where id=10;
> 查看事务
> trx_rows_locked: 3
> 与描述的不是期待的4行
> 使用相同表结构和数据，trx_rows_locked是5个，而不是期待的4个。
> 或者是我的期待有问题？
> mysql的版本是5.6.21

抱歉，在设置事务隔离级的时候出现问题。实际的事务隔离级为rr

60. 
闫宗帅
4月 16th, 201522:33
回复 | 引用 | #141

登博，您好，我想请教下：在铺助索引加锁后，对应的聚集索引也需要加锁，这情况在lock in share mode也适用吗？
我测试时，for update的话，在对应聚集索引加锁，但是lock in share mode的话，没有加

hedengcheng
4月 16th, 201522:38
回复 | 引用 | #142

聚簇索引记录上，一定会加锁的。

闫宗帅
9月 18th, 201523:05
回复 | 引用 | #143

您好，
测试的这种情况for update对应聚集索引加了锁，但是lock in share mode没加

1、表结构：
CREATE TABLE `c` (
`id1` int(11) NOT NULL DEFAULT ‘0’,
`id2` int(11) DEFAULT NULL,
`id3` int(11) DEFAULT NULL,
PRIMARY KEY (`id1`),
KEY `id2` (`id2`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
2、数据
MariaDB [mytest]> select *from c;
+——+——+——+
| id1 | id2 | id3 |
+——+——+——+
6	1	2
7	2	5
8	3	5
9	4	5
10	5	5
+——+——+——+
5 rows in set (0.00 sec)
3、对应执行计划：
MariaDB [mytest]> explain select id1 from c where id2 explain select id1 from c where id2 begin;
Query OK, 0 rows affected (0.00 sec)

MariaDB [mytest]> select id1 from c where id2 begin;
Query OK, 0 rows affected (0.00 sec)

MariaDB [mytest]> select id1 from c where id2<2 lock in share mode;
+——+
| id1 |
+——+
| 6 |
+——+
1 row in set (0.00 sec)

—TRANSACTION 284984, ACTIVE 6 sec
2 lock struct(s), heap size 320, 2 row lock(s)
MySQL thread id 2, OS thread handle 0xa61ffb90, query id 111 localhost root cleaning up
TABLE LOCK table `mytest`.`c` trx id 284984 lock mode IS
RECORD LOCKS space id 115 page no 4 n bits 80 index `id2` of table `mytest`.`c` trx id 284984 lock mode S
Record lock, heap no 2 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
0: len 4; hex 80000001; asc ;;
1: len 4; hex 80000006; asc ;;

Record lock, heap no 7 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
0: len 4; hex 80000002; asc ;;
1: len 4; hex 80000007; asc ;;

61. 林
5月 21st, 201516:11
回复 | 引用 | #144

登博主: 以下一段解释和结论是不是搞反结论了?(RR不存在幻读,那么幻读是会有幻象的吧) . 文章::: 所谓幻读，就是同一个事务，连续做两次当前读 (例如：select * from t1 where id = 10 for update;)，那么这两次当前读返回的是完全相同的记录 (记录数量一致，记录本身也一致)，第二次的当前读，不会比第一次返回更多的记录 (幻象)。

62. 西楼
6月 1st, 201522:04

回复 | 引用 | #145

困惑了！按你这分析加行锁是啥概念？直接加行锁不就结了，你这才两字段，这表要多几个带索引字段，是不可以通过各种where 字段来修改表，那如第二种情况，是不每个字段都要加x锁

63. 
Richard
6月 11th, 201510:36
回复 | 引用 | #146

大师，请问怎么解释这个：
CREATE TABLE test (a int, index (a));
INSERT INTO test VALUES (5), (10), (15);
RR级别下
tx1:
select * from test where a=10 for update;

tx2:
INSERT INTO test VALUES(5); //On hold
INSERT INTO test VALUES(9); //On hold
INSERT INTO test VALUES(14); //On hold
INSERT INTO test VALUES(4); //Works
INSERT INTO test VALUES (15); //Works
UPDATE test SET a = 1 WHERE a = 5; //Works
UPDATE test SET a = 8 WHERE a = 5; //On hold
UPDATE test SET a = 7 WHERE a = 15; //On hold
UPDATE test SET a = 100 WHERE a = 15; //Works

按分析，这种非唯一辅助索引，锁范围应该是(5,15)
奇怪的是：
INSERT INTO test VALUES(5); //On hold
UPDATE test SET a = 1 WHERE a = 5; //Works

5这个到底锁没有啊？

64. 
Richard
6月 11th, 201510:56
回复 | 引用 | #147

何大师，顺便再多问一个

关于那个Intention Lock, 具体是个什么鬼。
什么时候锁，什么时候起作用

看官方文档说得不清不楚的

65. 
jjbond
6月 23rd, 201515:22
回复 | 引用 | #148

大牛，打扰～我想咨询一下，如果是多字段的唯一索引（ex unique key(a,b)），那么RR + 唯一索引是不是也应该会产生gap锁?

- 
jjbond
6月 23rd, 201515:29
回复 | 引用 | #149

补充一下，delete操作只针对a 例如 delete…where a = xxx

66. Mysql文章笔记 | 程序员之家
7月 26th, 201520:59
#150
67. 
王三
9月 5th, 201512:13
回复 | 引用 | #151

非常nice的文章，授人以渔，解决了很多关于mysql锁的疑惑，感谢博主

68. 
闫宗帅
9月 18th, 201523:09
回复 | 引用 | #152

您好，
测试的这种情况for update对应聚集索引加了锁，但是lock in share mode没加

1、表结构：
CREATE TABLE `c` (
`id1` int(11) NOT NULL DEFAULT '0',
`id2` int(11) DEFAULT NULL,
`id3` int(11) DEFAULT NULL,
PRIMARY KEY (`id1`),
KEY `id2` (`id2`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
2、数据
MariaDB [mytest]> select *from c;
+——+——+——+
| id1 | id2 | id3 |
+——+——+——+
6	1	2
7	2	5
8	3	5
9	4	5
10	5	5
+——+——+——+
5 rows in set (0.00 sec)
3、对应执行计划：
MariaDB [mytest]> explain select id1 from c where id2 explain select id1 from c where id2 begin;
Query OK, 0 rows affected (0.00 sec)

MariaDB [mytest]> select id1 from c where id2 begin;
Query OK, 0 rows affected (0.00 sec)

MariaDB [mytest]> select id1 from c where id2<2 lock in share mode;
+——+
| id1 |
+——+
| 6 |
+——+
1 row in set (0.00 sec)

—TRANSACTION 284984, ACTIVE 6 sec
2 lock struct(s), heap size 320, 2 row lock(s)
MySQL thread id 2, OS thread handle 0xa61ffb90, query id 111 localhost root cleaning up
TABLE LOCK table `mytest`.`c` trx id 284984 lock mode IS
RECORD LOCKS space id 115 page no 4 n bits 80 index `id2` of table `mytest`.`c` trx id 284984 lock mode S
Record lock, heap no 2 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
0: len 4; hex 80000001; asc ;;
1: len 4; hex 80000006; asc ;;

Record lock, heap no 7 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
0: len 4; hex 80000002; asc ;;
1: len 4; hex 80000007; asc ;;

- 
  闫宗帅
  9月 18th, 201523:14
  回复 | 引用 | #153

  提交了2次都没显示全。

  我再提下加锁情况：
  1、select id1 from c where id2<2 for update;有4个锁，2个二级索引上锁，2个二级索引对应聚集索引上的锁
  3 lock struct(s), heap size 320, 4 row lock(s)
  MySQL thread id 2, OS thread handle 0xa61ffb90, query id 107 localhost root cleaning up
  TABLE LOCK table `mytest`.`c` trx id 284983 lock mode IX
  RECORD LOCKS space id 115 page no 4 n bits 80 index `id2` of table `mytest`.`c` trx id 284983 lock_mode X
  Record lock, heap no 2 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
  0: len 4; hex 80000001; asc ;;
  1: len 4; hex 80000006; asc ;;

  Record lock, heap no 7 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
  0: len 4; hex 80000002; asc ;;
  1: len 4; hex 80000007; asc ;;
  RECORD LOCKS space id 115 page no 3 n bits 88 index `PRIMARY` of table `mytest`.`c` trx id 284983 lock_mode X locks rec but not gap
  Record lock, heap no 7 PHYSICAL RECORD: n_fields 5; compact format; info bits 0
  0: len 4; hex 80000006; asc ;;
  1: len 6; hex 00000004577e; asc W~;;

2: len 7; hex 44000002190d34; asc D 4;;
3: len 4; hex 80000001; asc ;;
4: len 4; hex 80000002; asc ;;

Record lock, heap no 8 PHYSICAL RECORD: n_fields 5; compact format; info bits 0
0: len 4; hex 80000007; asc ;;
1: len 6; hex 000000045912; asc Y ;;
2: len 7; hex 100000014116d6; asc A ;;
3: len 4; hex 80000002; asc ;;
4: len 4; hex 80000005; asc ;;

2、select id1 from c where id2<2 lock in share mode;
2 lock struct(s), heap size 320, 2 row lock(s)
MySQL thread id 2, OS thread handle 0xa61ffb90, query id 111 localhost root cleaning up
TABLE LOCK table `mytest`.`c` trx id 284984 lock mode IS
RECORD LOCKS space id 115 page no 4 n bits 80 index `id2` of table `mytest`.`c` trx id 284984 lock mode S
Record lock, heap no 2 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
0: len 4; hex 80000001; asc ;;
1: len 4; hex 80000006; asc ;;

Record lock, heap no 7 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
0: len 4; hex 80000002; asc ;;
1: len 4; hex 80000007; asc ;;

69. MYSQL的锁机制 | Daniel Hu的技术博客
10月 4th, 201508:38
#154

70. 
Hereicome
10月 7th, 201521:00
回复 | 引用 | #155

之前片断式的看过一些锁的东西，但是还是有很多困惑，这篇文章解了很多困惑:)

71. 
dblover
10月 10th, 201517:53
回复 | 引用 | #156

下面几个是关键信息啊：
当前读：特殊的读操作，插入/更新/删除操作，属于当前读，需要加锁。
当前读，读取的是记录的最新版本，必须保证是已经提交的一致版本。
select * from table where ? lock in share mode;
select * from table where ? for update;
insert into table values (...);
update table set ? where ?;
delete from table where ?;

为什么将 插入/更新/删除 操作，都归为当前读？
针对一条当前读的SQL语句，InnoDB与MySQL Server的交互，是一条一条进行的，因此，加锁也是一条一条进行的。先对一条满足条件的记录加锁，返回给MySQL Server，做一些DML操作；然后在读取下一条加锁，直至读取完毕。

72. 
lexin
10月 28th, 201514:26
回复 | 引用 | #157

> **hedengcheng** :谁说Intention Lock是表级锁？谁写的？拉出去突突了。

http://dev.mysql.com/doc/refman/5.5/en/innodb-lock-modes.html

Intention Locks

Additionally, InnoDB supports multiple granularity locking which permits coexistence of record locks and locks on entire tables. To make locking at multiple granularity levels practical, additional types of locks called intention locks are used. Intention locks are table locks in InnoDB that indicate which type of lock (shared or exclusive) a transaction will require later for a row in that table. There are two types of intention locks used in InnoDB

73. 
江河汇
11月 2nd, 201516:10
回复 | 引用 | #158

针对SQL：select * from t1 where id = 10 for update; 第一次查询，没有找到满足查询条件的记录，那么GAP锁是否还能够省略？此问题留给大家思考。

这个不需要了吧。在rr隔离级别下， 第一次查询一般新建readview，假如有其他的事物插入了一条，在进行读取的时候能够通过readview机制，读不到新的事物插入的记录。

74. liuyingjie
11月 9th, 201517:44
回复 | 引用 | #159

何老师，我的生产环境中时长出现子查询的情况，而子查询又往往与join操作联系在一起

当join操作之后，又使用where 条件进行过滤时，请问加锁是根据join操作中选择的字段呢 还是 根据where条件中的字段

75. chengweiming
11月 16th, 201511:00
回复 | 引用 | #160

登博，辛苦帮忙看看。
验证加锁复杂SQL的实践：

MYSQL 5.6,ICP在使用中遇到LOCK的问题，求解惑，详情见下文：
环境：
数据库版本：mysql Ver 14.14 Distrib 5.6.25, for Linux (x86_64) using EditLine wrapper
数据库隔离级别： REPEATABLE-READ
autocommit ：OFF optimizer_switch：
index_merge=on,index_merge_union=on,index_merge_sort_union=on,index_merge_intersection=on,engine_condition_pushdown=on,
数据：
1、 表结构：
mysql> show create table t1\G
*************************** 1. row ***************************
Table: t1
Create Table: CREATE TABLE `t1` (
`a` int(11) NOT NULL,
`b` int(11) NOT NULL,
`c` int(11) NOT NULL,
`d` int(11) NOT NULL,
`e` varchar(20) DEFAULT NULL,
PRIMARY KEY (`a`),
KEY `idx_t1_bcd` (`b`,`c`,`d`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

2、表数据：
mysql> select * from t1;
+—+—+—+—+——+
| a | b | c | d | e |
+—+—+—+—+——+
1	1	1	1	a
2	2	2	2	b
3	3	2	2	c
4	3	1	1	d
5	2	3	5	e
6	6	4	4	f
7	4	5	5	g
8	8	8	8	h
+—+—+—+—+——+
8 rows in set (0.00 sec)

操作过程：
session 1:
delete from t1 where b>2 and b explain select * from t1 where b>2 and b select * from information_schema.innodb_locks;+
——————+—————-+—————+—————+—————-+—————+—————+—————+————-+—————+
| lock_id | lock_trx_id | lock_mode | lock_type | lock_table | lock_index | lock_space | lock_page | lock_rec | lock_data |
+——————+—————-+—————+—————+—————-+—————+—————+—————+————-+—————+
| 38777:390:3:5 | 38777 | X | RECORD | `test`.`t1` | PRIMARY | 390 | 3 | 5 | 4 |
| 38771:390:3:5 | 38771 | X | RECORD | `test`.`t1` | PRIMARY | 390 | 3 | 5 | 4 |
+——————+—————-+—————+—————+—————-+—————+—————+—————+————-+—————+

根据锁及ICP的知识，此时加锁的情况应该是在索引 idx_t1_bcd 上的b>2 and b<5之间加gap lock,
idx_t1_bcd 上的c=2 加 X锁
主键 a=3 加 x 锁。
应该a=4上是没有加X锁的，可以进行删除与更改。
但是从session3上的结果来，此时a=4上被加上了X锁。

添加lock monitor,加锁详情如下：
设置：set global innodb_status_output_locks=on

show engine innodb status:

—TRANSACTION 39948, ACTIVE 15 sec
3 lock struct(s), heap size 360, 8 row lock(s), undo log entries 1
MySQL thread id 1, OS thread handle 0x7f7f44263700, query id 15 127.0.0.1 root cleaning up
TABLE LOCK table `test`.`t1` trx id 39948 lock mode IX
RECORD LOCKS space id 390 page no 4 n bits 80 index `idx_t1_bcd` of table `test`.`t1` trx id 39948 lock_mode X
Record lock, heap no 5 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
0: len 4; hex 80000003; asc ;;
1: len 4; hex 80000001; asc ;;
2: len 4; hex 80000001; asc ;;
3: len 4; hex 80000004; asc ;;

Record lock, heap no 6 PHYSICAL RECORD: n_fields 4; compact format; info bits 32
0: len 4; hex 80000003; asc ;;
1: len 4; hex 80000002; asc ;;
2: len 4; hex 80000002; asc ;;
3: len 4; hex 80000003; asc ;;

Record lock, heap no 7 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
0: len 4; hex 80000004; asc ;;
1: len 4; hex 80000005; asc ;;
2: len 4; hex 80000005; asc ;;
3: len 4; hex 80000007; asc ;;

Record lock, heap no 8 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
0: len 4; hex 80000006; asc ;;
1: len 4; hex 80000004; asc ;;
2: len 4; hex 80000004; asc ;;
3: len 4; hex 80000006; asc ;;

RECORD LOCKS space id 390 page no 3 n bits 80 index `PRIMARY` of table `test`.`t1` trx id 39948 lock_mode X locks rec but not gap
Record lock, heap no 4 PHYSICAL RECORD: n_fields 7; compact format; info bits 32
0: len 4; hex 80000003; asc ;;
1: len 6; hex 000000009c0c; asc ;;
2: len 7; hex 0b000001e20224; asc $;;
3: len 4; hex 80000003; asc ;;
4: len 4; hex 80000002; asc ;;
5: len 4; hex 80000002; asc ;;
6: len 1; hex 63; asc c;;

Record lock, heap no 5 PHYSICAL RECORD: n_fields 7; compact format; info bits 0
0: len 4; hex 80000004; asc ;;
1: len 6; hex 000000009770; asc p;;
2: len 7; hex 200000026e0e30; asc n 0;;
3: len 4; hex 80000003; asc ;;
4: len 4; hex 80000001; asc ;;
5: len 4; hex 80000001; asc ;;
6: len 1; hex 64; asc d;;

Record lock, heap no 7 PHYSICAL RECORD: n_fields 7; compact format; info bits 0
0: len 4; hex 80000006; asc ;;
1: len 6; hex 00000000970e; asc ;;
2: len 7; hex dd000001eb0110; asc ;;
3: len 4; hex 80000006; asc ;;
4: len 4; hex 80000004; asc ;;
5: len 4; hex 80000004; asc ;;
6: len 1; hex 66; asc f;;

Record lock, heap no 8 PHYSICAL RECORD: n_fields 7; compact format; info bits 0
0: len 4; hex 80000007; asc ;;
1: len 6; hex 00000000970d; asc ;;
2: len 7; hex dc000001710110; asc q ;;
3: len 4; hex 80000004; asc ;;
4: len 4; hex 80000005; asc ;;
5: len 4; hex 80000005; asc ;;
6: len 1; hex 67; asc g;;

76. 
chengweiming
11月 16th, 201511:08
回复 | 引用 | #161

上面的操作过程显示乱了，我重新贴下：

操作过程：
session 1:
delete from t1 where b>2 and b explain select * from t1 where b>2 and b select * from information_schema.innodb_locks;+
————————+————————-+————————+————————+————————-+————————+————————+————————+————————-+————————+
| lock_id | lock_trx_id | lock_mode | lock_type | lock_table | lock_index | lock_space | lock_page | lock_rec | lock_data |
+————————+————————-+————————+————————+————————-+————————+————————+————————+————————-+————————+
| 38777:390:3:5 | 38777 | X | RECORD | `test`.`t1` | PRIMARY | 390 | 3 | 5 | 4 |
| 38771:390:3:5 | 38771 | X | RECORD | `test`.`t1` | PRIMARY | 390 | 3 | 5 | 4 |
+————————+————————-+————————+————————+————————-+————————+————————+————————+————————-+————————+

根据锁及ICP的知识，此时加锁的情况应该是在索引 idx_t1_bcd 上的b>2 and b<5之间加gap lock,
idx_t1_bcd 上的c=2 加 X锁
主键 a=3 加 x 锁。
应该a=4上是没有加X锁的，可以进行删除与更改。
但是从session3上的结果来，此时a=4上被加上了X锁。

77. 
chengweiming
11月 16th, 201511:11
回复 | 引用 | #162

贴上去就显示错乱了，不好意思。
session 1:
‘delete from t1 where b>2 and b<5 and c=2;’

78. 
chengweiming
11月 16th, 201511:12
回复 | 引用 | #163

session 2:
delete from t1 where a=4
————被锁住

79. 
chengweiming
11月 16th, 201511:12
回复 | 引用 | #164

session 3:
mysql> select * from information_schema.innodb_locks;+————————+————————-+————————+————————+————————-+————————+————————+————————+————————-+————————+
+————————+————————-+————————+————————+————————-+————————+————————+————————+————————-+————————+
| lock_id | lock_trx_id | lock_mode | lock_type | lock_table | lock_index | lock_space | lock_page | lock_rec | lock_data |
+————————+————————-+————————+————————+————————-+————————+————————+————————+————————-+————————+
| 38777:390:3:5 | 38777 | X | RECORD | `test`.`t1` | PRIMARY | 390 | 3 | 5 | 4 |
| 38771:390:3:5 | 38771 | X | RECORD | `test`.`t1` | PRIMARY | 390 | 3 | 5 | 4 |
+————————+————————-+————————+————————+————————-+————————+————————+————————+————————-+————————+

80. 关于mysql锁的学习 | PHPor 的Blog
12月 25th, 201515:15
#165

81. 
chinaxing
1月 5th, 201611:34
回复 | 引用 | #166

博主好，我这边也遇到一个比较费解的mysql 死锁。根据你的文章自己思考了没分析出来。帮忙分析下，多谢了：）

http://stackoverflow.com/questions/34593744/why-insert-statement-hold-gap-lock-in-mysql-innodb

82. 
welyss
1月 8th, 201614:30
回复 | 引用 | #167

这是我见过的，mysql最棒的博文了！非常浅显易懂，感谢lz分享

83. 
小马哥
2月 3rd, 201616:36
回复 | 引用 | #168

非常不错

84. mysql死锁问题分析-IT大道
2月 5th, 201616:33

#169

85. MySQL 加锁处理分析 | 王二宝的小站
3月 10th, 201618:06
#170
86. MySQL/InnoDB 加锁处理分析 | phper
3月 11th, 201613:44
#171
87. MySQL 死锁问题分析 – 码农网
4月 1st, 201615:59
#172
88. 
书生
4月 7th, 201621:35
回复 | 引用 | #173

请教博主个问题，在组合七的情况下，如果一个session执行:
select * from myt where id>10 for update;
而第二个session插入记录：
insert into myt values(‘bb’, 10);
第二个session的插入操作会卡住，求教下第一个语句加的锁是哪些？

89. 
书生
4月 7th, 201621:36
回复 | 引用 | #174

请教博主个问题，在组合七的情况下，如果一个session执行:
select * from T1 where id>10 for update;
而第二个session插入记录：
insert into T1 values(‘bb’, 10);
第二个session的插入操作会卡住，求教下第一个语句加的锁是哪些？

90. 
gameanimal
4月 23rd, 201613:40
回复 | 引用 | #175

何大师您好：
最近工作中使用mysql的过程中遇到一个死锁，
mysql version 5.5.20
事务隔离级别是read committed
表结构
create table if not exists ‘m’
(
aid bigint not null AUTO_INCREMENT,
id bigint not null default -1,
sendguid bigint not null default -1,
sendname varchar(80) binary not null,
writetime bigint not null default 0,
receiveguid bigint not null default -1,
readtime bigint not null default 0,
mailtype smallint not null default 0,
content varchar(192) not null,
moneytype smallint not null default 0,
moneyvalue int not null default 0,
boxtype tinyint not null default 0,
isvalid tinyint not null default 0,
itemguid bigint not null default -1,
dataid int not null default -1,
binded tinyint not null default -1,
stackcount smallint not null default -1,
createtime bigint not null default -1,
enchancelevel smallint not null default -1,
enchanceexp int not null default -1,
enchancetotalexp bigint not null default -1,
starlevel smallint not null default -1,
startimes smallint not null default -1,
dynamicdata1 int not null default -1,
dynamicdata2 int not null default -1,
dynamicdata3 int not null default -1,
dynamicdata4 int not null default -1,
dynamicdata5 int not null default -1,
dynamicdata6 int not null default -1,
dynamicdata7 int not null default -1,

```
dynamicdata8 int not null default -1,
origin int not null default 0,
primary key (aid)
)ENGINE = INNODB;

create unique index Index_mail_guid on t_usermail
(
guid,
boxtype
);
create index Index_mail_charguid on t_usermail
(
receiveguid,
boxtype
);
```

91. 970807800@qq.com
6月 15th, 201614:19
回复 | 引用 | #176

博主好，我在我本地测试第二个死锁例子时，只会报” ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transacti” ，并没有死锁，本人对锁机制不是很清楚，能帮忙解惑吗？

92. fool
7月 20th, 201619:24
回复 | 引用 | #177

?

93. mysql事务和锁InnoDB_阿里欧歌
8月 7th, 201616:05
#178

昵称 (必填)
电子邮箱 (我们会为您保密) (必填)
网址

订阅评论 提交评论

Recent Posts

- 2017阿里巴巴云栖大会数据库内核专场，内容概要
- 阿里巴巴数据库内核团队简介&纳新
- 生活中的Paxos，原来你我都在使用——对Paxos生活化的解读
- 数据一致性-分区可用性-性能——多副本强同步数据库系统实现之我见
- 一个最不可思议的MySQL死锁分析

Tag Cloud

2PC 5.6 add index AIO B+-Tree Block Buffer Pool Checkpoint CPU Crash Recovery Database Data Cache Deadlock Group Commit InnoDB Insert Buffer join Linux Lock MariaDB MVCC MySQL MySQL BUG online Oracle Page Percona Range Query ReadView Redo Rollback Segment Row transaction Trasaction Undo XA XtraDB 中断 分布式 分布式事务 多版本 数据库 数据库内核分享 日志 软中断

Categories

- Architecture (1)
- C/C++ (2)
- Distributed (2)
- Falcon (1)
- Hardware (3)
- Optimizer (2)
- Performance (1)
- Programming (7)
- Test (1)
- TPCC (1)
- 数据库 (45)
  - InnoDB (34)

- MySQL Server (27)
- Oracle (6)
- PostgreSQL (1)

- 数据库内核分享 (14)
- 数据库研发 (6)
- 杂谈 (3)

Archives

- 2017年九月
- 2017年三月
- 2016年五月
- 2015年三月
- 2014年一月
- 2013年十二月
- 2013年十一月
- 2013年七月
- 2013年四月
- 2013年三月
- 2013年二月
- 2013年一月
- 2012年十二月
- 2012年十一月
- 2012年十月
- 2012年九月
- 2012年八月
- 2012年七月
- 2012年六月
- 2012年五月
- 2012年四月

Blogroll

- ACM Queue
- All Things Distributed
- Bartosz Milewski's Programming Cafe
- Brendan's blog
- Dimitrik's Weblog
- Dr.Dobb's
- Gustavo Duarte
- High Scalability
- Jeremy Cole's Blog
- Jonathan Lewis's Oracle Scratchpad
- LinkedIn Engineering
- Mark Callaghan's High Availability MySQL
- Mechanical Sympathy
- MySQL Performance Blog
- Paul E. McKenney's Journal
- Perspectives
- preshing on programming
- Sutter’ s Mill
- The Pith of Performance
- The Twitter Engineering Blog
- Transactions on InnoDB
- Wired Enterprise – IT Happens
- 云风的Blog
- 左耳朵耗子的酷壳
- 旺旺的数据库思考者之家
- 霸爷的系统技术非业余研究
- 首席的弯曲评论

- 注册
- 登录
- 置顶