* --ASP丶NET

**[mysql（四）------Mysql中的锁](https://www.cnblogs.com/flgb/p/12154377.html)**

Posted on 2020-01-05 23:43  [FLGB](https://www.cnblogs.com/flgb/)  阅读(9)  评论(0)  [编辑](https://i.cnblogs.com/EditPosts.aspx?postid=12154377)  [收藏](javascript:void(0))

1.

2 MySQL InnoDB 锁的基本类型

https://dev.mysql.com/doc/refman/5.7/en/innodb-locking.html

官网把锁分成了 8 类。所以我们把前面的两个行级别的锁（Shared and Exclusive

Locks），和两个表级别的锁（Intention Locks）称为锁的基本模式。

后面三个 Record Locks、Gap Locks、Next-Key Locks，我们把它们叫做锁的算法，

也就是分别在什么情况下锁定什么范围。

2.1 锁的粒度

我们讲到 InnoDB 里面既有行级别的锁，又有表级别的锁，我们先来分析一下这两

种锁定粒度的一些差异。

表锁，顾名思义，是锁住一张表；行锁就是锁住表里面的一行数据。锁定粒度，表

锁肯定是大于行锁的。

那么加锁效率，表锁应该是大于行锁还是小于行锁呢？大于。为什么？表锁只需要

直接锁住这张表就行了，而行锁，还需要在表里面去检索这一行数据，所以表锁的加锁

效率更高。

第二个冲突的概率？表锁的冲突概率比行锁大，还是小？

大于，因为当我们锁住一张表的时候，其他任何一个事务都不能操作这张表。但是

我们锁住了表里面的一行数据的时候，其他的事务还可以来操作表里面的其他没有被锁

定的行，所以表锁的冲突概率更大。

表锁的冲突概率更大，所以并发性能更低，这里并发性能就是小于。

nnoDB 里面我们知道它既支持表锁又支持行锁，另一个常用的存储引擎 MyISAM 支

持什么粒度的锁？这是第一个问题。第二个就是 InnoDB 已经支持行锁了，那么它也可以通过把表里面的每一行都锁住来实现表锁，为什么还要提供表锁呢？

要搞清楚这个问题，我们就要来了解一下 InnoDB 里面的基本的锁的模式（lock

mode），这里面有两个行锁和两个表锁。

2.2 共享锁

第一个行级别的锁就是我们在官网看到的 Shared Locks （共享锁），我们获取了

一行数据的读锁以后，可以用来读取数据，所以它也叫做读锁，注意不要在加上了读锁

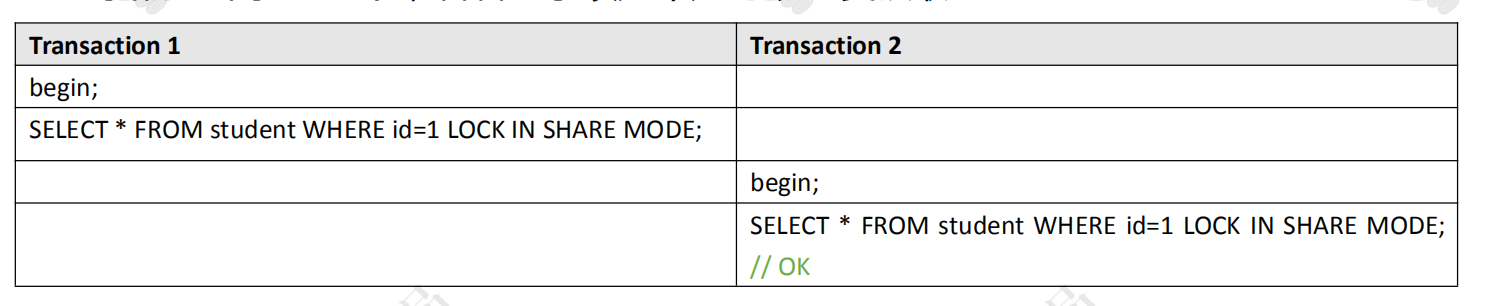
以后去写数据，不然的话可能会出现死锁的情况。而且多个事务可以共享一把读锁。那

怎么给一行数据加上读锁呢？

我们可以用 select …… lock in share mode; 的方式手工加上一把读锁。

释放锁有两种方式，只要事务结束，锁就会自动事务，包括提交事务和结束事务。

我们也来验证一下，看看共享锁是不是可以重复获取。



 Select – in lock share mode

2.3 排它锁

第二个行级别的锁叫做 Exclusive Locks（排它锁），它是用来操作数据的，所以又

叫做写锁。只要一个事务获取了一行数据的排它锁，其他的事务就不能再获取这一行数据的共享锁和排它锁。

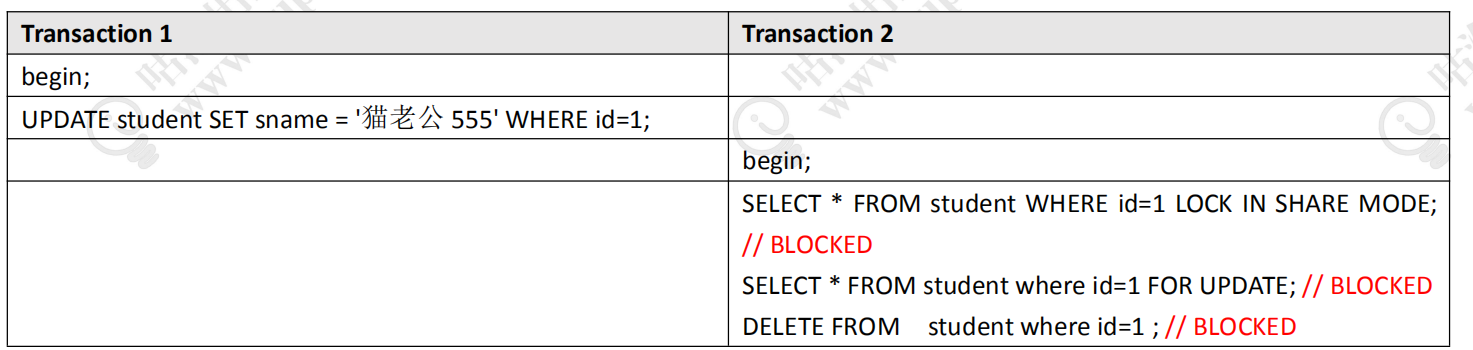
排它锁的加锁方式有两种，第一种是自动加排他锁。我们在操作数据的时候，包括

增删改，都会默认加上一个排它锁。

还有一种是手工加锁，我们用一个 FOR UPDATE 给一行数据加上一个排它锁，这个无论是在我们的代码里面还是操作数据的工具里面，都比较常用。

释放锁的方式跟前面是一样的。

排他锁的验证：



这个是两个行锁，接下来就是两个表锁。

2.4 意向锁

意向锁是什么呢？我们好像从来没有听过，也从来没有使用过，其实他们是由数据

库自己维护的。

也就是说，当我们给一行数据加上共享锁之前，数据库会自动在这张表上面加一个

**意向共享锁**。

当我们给一行数据加上排他锁之前，数据库会自动在这张表上面加一个**意向排他锁**。

反过来说：

如果一张表上面至少有一个意向共享锁，说明有其他的事务给其中的某些数据行加

上了共享锁。

如果一张表上面至少有一个意向排他锁，说明有其他的事务给其中的某些数据行加

上了排他锁

select \* from t2 where id =4 for update;

TABLE LOCK table `gupao`.`t2` trx id 24467 lock mode IX

RECORD LOCKS space id 64 page no 3 n bits 72 index PRIMARY of table `gupao`.`t2` trx id 24467 lock\_mode X locks rec but not

gap

那么这两个表级别的锁存在的意义是什么呢？第一个，我们有了表级别的锁，在

InnoDB 里面就可以支持更多粒度的锁。它的第二个作用，我们想一下，如果说没有意向

锁的话，当我们准备给一张表加上表锁的时候，我们首先要做什么？是不是必须先要去判断有没其他的事务锁定了其中了某些行？如果有的话，肯定不能加上表锁。那么这个

时候我们就要去扫描整张表才能确定能不能成功加上一个表锁，如果数据量特别大，比

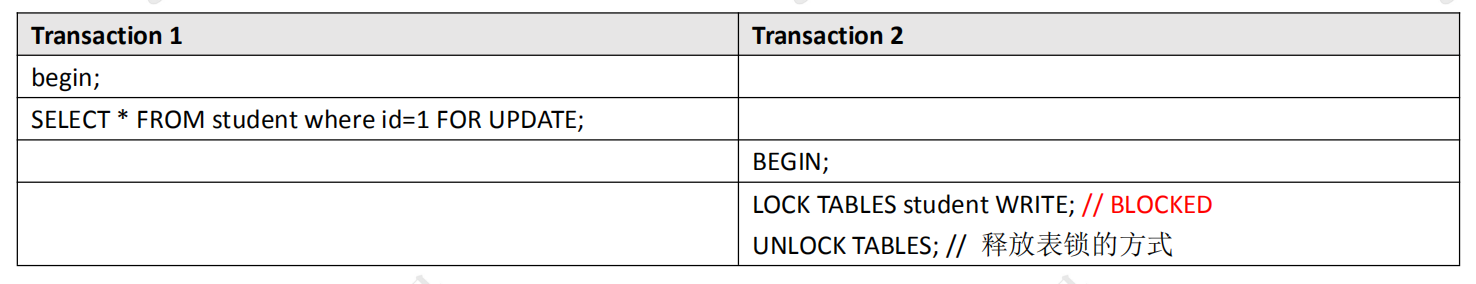
如有上千万的数据的时候，加表锁的效率是不是很低？

但是我们引入了意向锁之后就不一样了。我只要判断这张表上面有没有意向锁，如

果有，就直接返回失败。如果没有，就可以加锁成功。所以 InnoDB 里面的表锁，我们

可以把它理解成一个标志。就像火车上厕所有没有人使用的灯，是用来提高加锁的效率

的。



以上就是 MySQL 里面的 4 种基本的锁的模式，或者叫做锁的类型。

到这里我们要思考两个问题，首先，锁的作用是什么？它跟 Java 里面的锁是一样的，

是为了解决资源竞争的问题，Java 里面的资源是对象，数据库的资源就是数据表或者数

据行。

所以**锁是用来解决事务对数据的并发访问的问题的**。

那么，锁到底锁住了什么呢？

当一个事务锁住了一行数据的时候，其他的事务不能操作这一行数据，那它到底是

锁住了这一行数据，还是锁住了这一个字段，还是锁住了别的什么东西呢？

3 行锁的原理

3.1 没有索引的表（假设锁住记录）

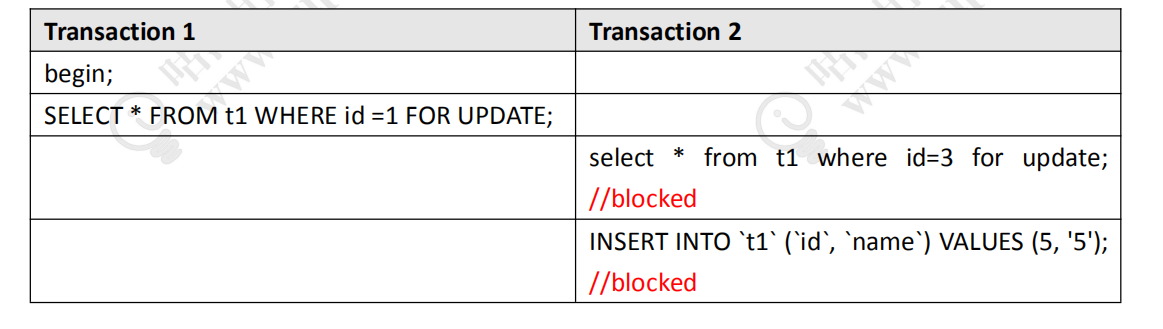
首先我们有三张表，一张没有索引的 t1，一张有主键索引的 t2，一张有唯一索引的t3。

我们先假设 InnoDB 的锁锁住了是一行数据或者一条记录。

我们先来看一下 t1 的表结构，它有两个字段，

int 类型的 id 和 varchar 类型的 name。

里面有 4 条数据，1、2、3、4。



现在我们在两个会话里面手工开启两个事务。

在第一个事务里面，我们通过 where id =1 锁住第一行数据。

在第二个事务里面，我们尝试给 id=3 的这一行数据加锁，大家觉得能成功吗？

【互动】觉得能成功刷 1，觉得不能成功的刷 0。

很遗憾，我们看到红灯亮起，这个加锁的操作被阻塞了。这就有点奇怪了，第一个

事务锁住了 id=1 的这行数据，为什么我不能操作 id=3 的数据呢？

我们再来操作一条不存在的数据，插入 id=5。它也被阻塞了。实际上这里整张表都

被锁住了。所以，我们的第一个猜想被推翻了，InnoDB 的锁锁住的应该不是 Record。

那为什么在没有索引或者没有用到索引的情况下，会锁住整张表？这个问题我们先

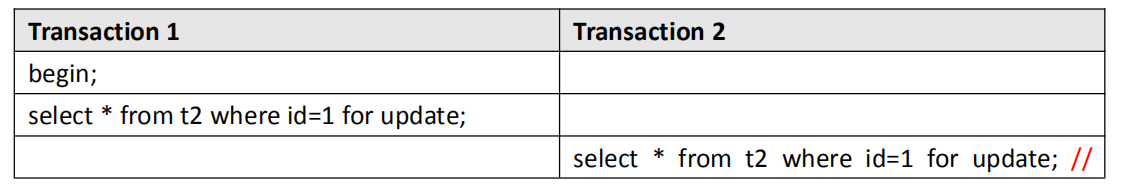
留在这里。

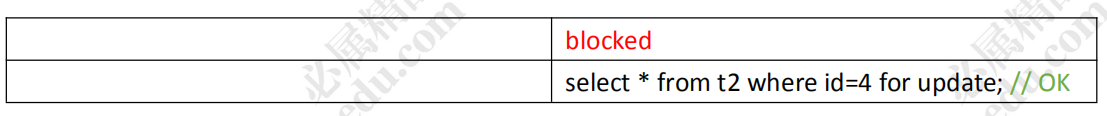
我们继续看第二个演示。

3.2 有主键索引的表

我们看一下 t2 的表结构。字段是一样的，不同的地方是 id 上创建了一个主键索引。

里面的数据是 1、4、7、10。





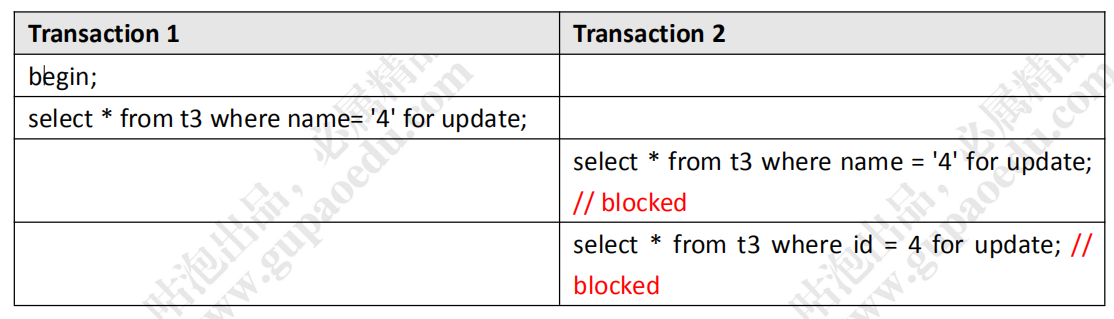
第一种情况，使用相同的 id 值去加锁，冲突；使用不同的 id 加锁，可以加锁成功。

那么，既然不是锁定一行数据，有没有可能是锁住了 id 的这个字段呢？

3.3 唯一索引（假设锁住字段）

我们看一下 t3 的表结构。字段还是一样的， id 上创建了一个主键索引，name 上

创建了一个唯一索引。里面的数据是 1、4、7、10。



在第一个事务里面，我们通过 name 字段去锁定值是 4 的这行数据。

在第二个事务里面，尝试获取一样的排它锁，肯定是失败的，这个不用怀疑。

在这里我们怀疑 InnoDB 锁住的是字段，所以这次我换一个字段，用 id=4 去给这行

数据加锁，大家觉得能成功吗？

【互动】觉得能成功的刷一波 1，觉得不能成功的刷一波 0。

很遗憾，又被阻塞了，说明锁住的是字段的这个推测也是错的，否则就不会出现第

一个事务锁住了 name，第二个字段锁住 id 失败的情况。

既然锁住的不是 record，也不是 column，InnoDB 里面锁住的到底是什么呢？在这

三个案例里面，我们要去分析一下他们的差异在哪里，也就是这三张表的结构，是什么

区别导致了加锁的行为的差异？其实答案就是索引。InnoDB 的行锁，就是通过锁住索引

来实现的。

那索引又是个什么东西？为什么它可以被锁住？我们在第二节课里面已经分析过

了。

那么我们还有两个问题没有解决：

1、为什么表里面没有索引的时候，锁住一行数据会导致锁表？

或者说，如果锁住的是索引，一张表没有索引怎么办？

所以，一张表有没有可能没有索引？

1）如果我们定义了主键(PRIMARY KEY)，那么 InnoDB 会选择主键作为聚集索引。

2）如果没有显式定义主键，则 InnoDB 会选择第一个不包含有 NULL 值的唯一索

引作为主键索引。

3）如果也没有这样的唯一索引，则 InnoDB 会选择内置 6 字节长的 ROWID 作为隐藏的聚集索引，它会随着行记录的写入而主键递增。

所以，为什么锁表，是因为查询没有使用索引，会进行全表扫描，然后把每一个隐藏的聚集索引都锁住了。

2、为什么通过唯一索引给数据行加锁，主键索引也会被锁住？

大家还记得在 InnoDB 里面，当我们使用辅助索引的时候，它是怎么检索数据的吗？

辅助索引的叶子节点存储的是什么内容？

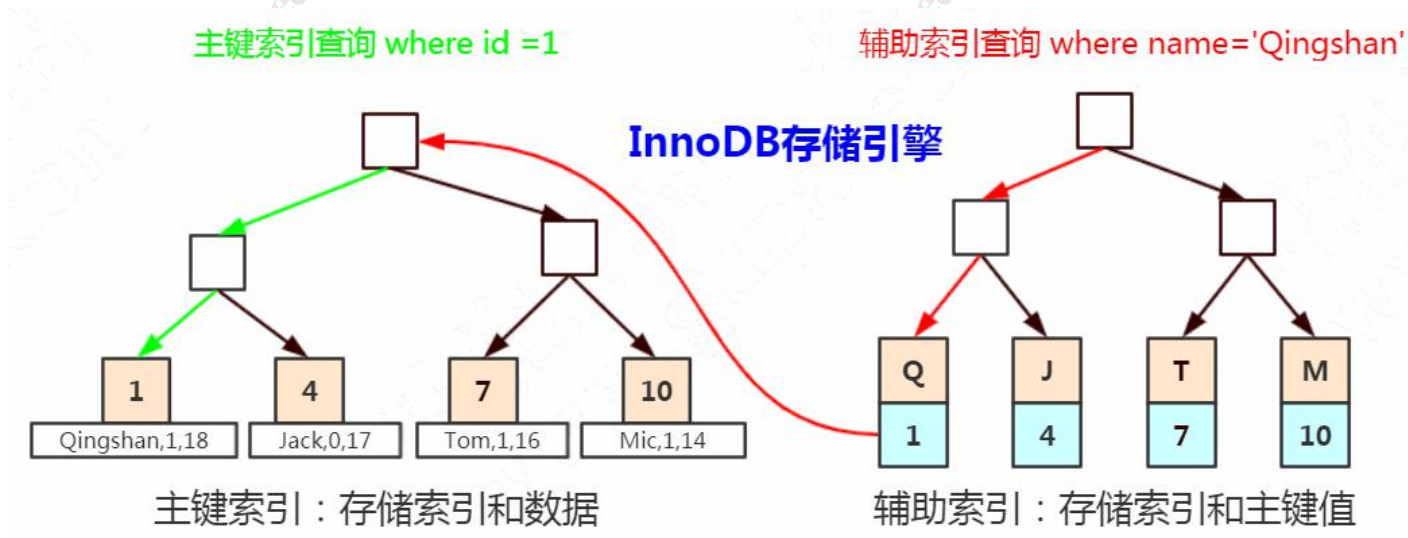
在辅助索引里面，索引存储的是二级索引和主键的值。比如name=4，存储的是name

的索引和主键 id 的值 4。

而主键索引里面除了索引之外，还存储了完整的数据。所以我们通过辅助索引锁定

一行数据的时候，它跟我们检索数据的步骤是一样的，会通过主键值找到主键索引，然

后也锁定。



现在我们已经搞清楚 4 个锁的基本类型和锁的原理了，在官网上，还有 3 种锁，我

们把它理解为锁的算法。我们也来看下 InnoDB 在什么时候分别锁住什么范围。

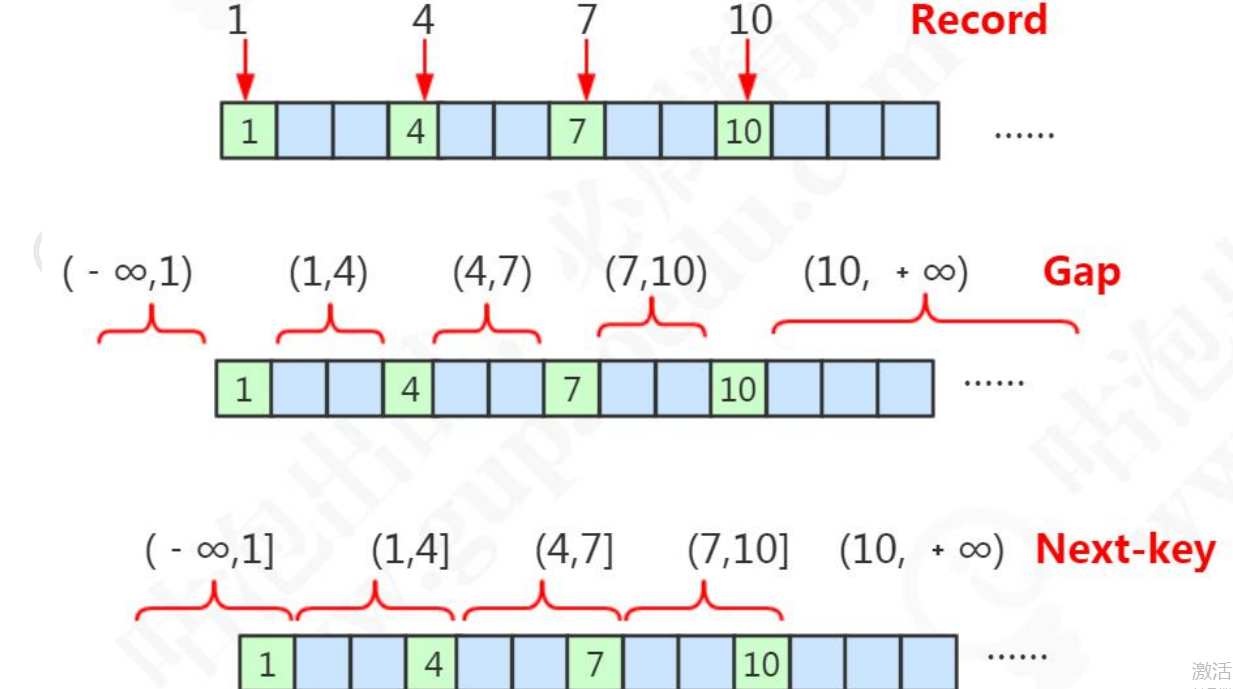
# 4 锁的算法

我们先来看一下我们测试用的表，t2，这张表有一个主键索引。

我们插入了 4 行数据，主键值分别是 1、4、7、10。

为了让大家真正理解这三种行锁算法的区别，我们需要了解一下三种范围的概念。

因为我们用主键索引加锁，我们这里的划分标准就是主键索引的值。



这些数据库里面存在的主键值，我们把它叫做 Record，记录，那么这里我们就有 4

个 Record。

根据主键，这些存在的 Record 隔开的数据不存在的区间，我们把它叫做 Gap，间隙，它是一个左开右开的区间。

最后一个，间隙（Gap）连同它左边的记录（Record），我们把它叫做临键的区间，

它是一个左开右闭的区间。

t2 的主键索引，它是整型的，可以排序，所以才有这种区间。如果我的主键索引不

是整形，是字符怎么办呢？字符可以排序吗？ 用 ASCII 码来排序。

我们已经弄清楚了三个范围的概念，下面我们就来看一下在不同的范围下，行锁是

怎么表现的。

4.1 记录锁

第一种情况，当我们对于唯一性的索引（包括唯一索引和主键索引）使用等值查询，

精准匹配到一条记录的时候，这个时候使用的就是记录锁。

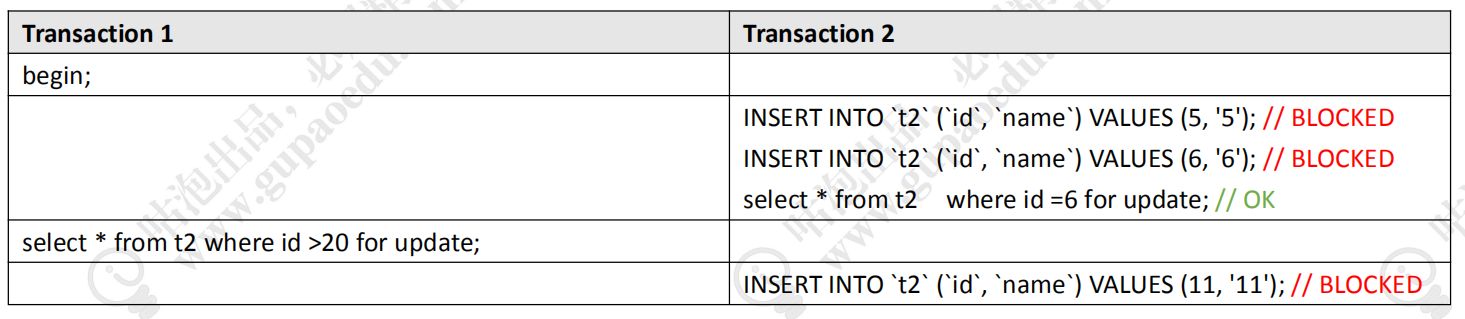
比如 where id = 1 4 7 10 。这个演示我们在前面已经看过了。我们使用不同的 key 去加锁，不会冲突，它只锁住这个 record。

4.2 间隙锁

第二种情况，当我们查询的记录不存在，没有命中任何一个 record，无论是用等值

查询还是范围查询的时候，它使用的都是间隙锁。

举个例子，where id >4 and id <7，where id = 6。



重复一遍，当查询的记录不存在的时候，使用间隙锁。

注意，**间隙锁主要是阻塞插入 insert**。相同的间隙锁之间不冲突。

Gap Lock 只在 RR 中存在。如果要关闭间隙锁，就是把事务隔离级别设置成 RC，

并且把 innodb\_locks\_unsafe\_for\_binlog 设置为 ON。

这种情况下除了外键约束和唯一性检查会加间隙锁，其他情况都不会用间隙锁。

4.3 临键锁

第三种情况，当我们使用了范围查询，不仅仅命中了 Record 记录，还包含了 Gap

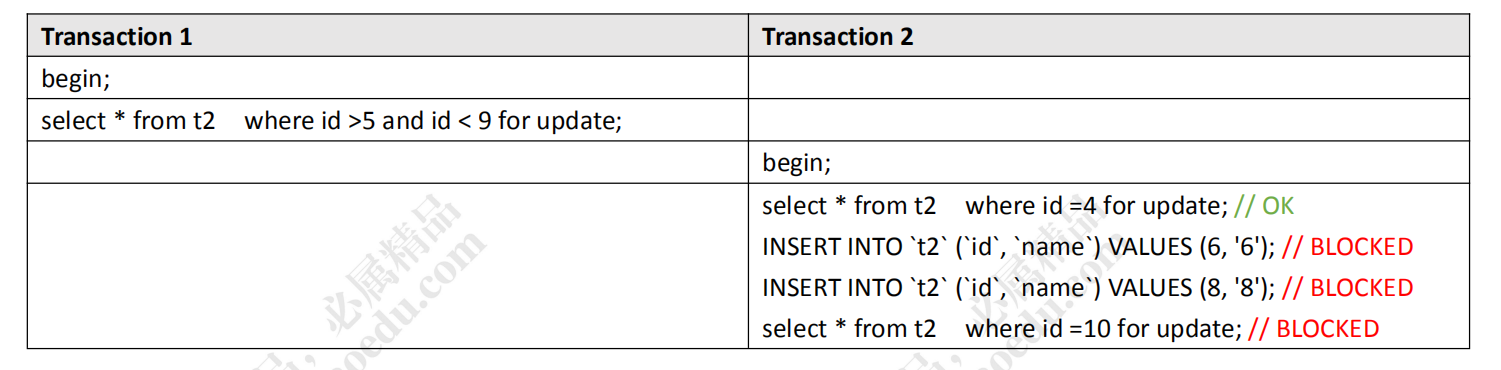
间隙，在这种情况下我们使用的就是临键锁，它是 MySQL 里面默认的行锁算法，相当于记录锁加上间隙锁。

其他两种退化的情况：

唯一性索引，等值查询匹配到一条记录的时候，退化成记录锁。

没有匹配到任何记录的时候，退化成间隙锁。

比如我们使用>5 <9， 它包含了记录不存在的区间，也包含了一个 Record 7。



临键锁，锁住最后一个 key 的下一个左开右闭的区间。

select \* from t2 where id >5 and id <=7 for update; -- 锁住(4,7]和(7,10]

select \* from t2 where id >8 and id <=10 for update; -- 锁住 (7,10]，(10,+∞)

为什么要锁住下一个左开右闭的区间？——就是为了解决幻读的问题。

4.4 小结：隔离级别的实现

所以，我们再回过头来看下这张图片，为什么 InnoDB 的 RR 级别能够解决幻读的

问题，就是用临键锁实现的。

我们再回过头来看下这张图片，这个就是MySQL InnoDB里面事务隔离级别的实现



最后我们来总结一下四个事务隔离级别的实现：

4.4.1 Read Uncommited

RU 隔离级别：不加锁。

4.4.2 Serializable

Serializable 所有的 select 语句都会被隐式的转化为 select ... in share mode，会

和 update、delete 互斥。

这两个很好理解，主要是 RR 和 RC 的区别？

4.4.3 Repeatable Read

RR 隔离级别下，普通的 select 使用快照读(snapshot read)，底层使用 MVCC 来实现。加锁的 select(select ... in share mode / select ... for update)以及更新操作

update, delete 等语句使用当前读（current read），底层使用记录锁、或者间隙锁、

临键锁。

4.4.4 Read Commited

RC 隔离级别下，普通的 select 都是快照读，使用 MVCC 实现。

加锁的 select 都使用记录锁，因为没有 Gap Lock。

除了两种特殊情况——外键约束检查(foreign-key constraint checking)以及重复

键检查(duplicate-key checking)时会使用间隙锁封锁区间。

所以 RC 会出现幻读的问题。

5 事务隔离级别怎么选？

https://dev.mysql.com/doc/refman/5.7/en/innodb-transaction-isolation-levels.html

RU 和 Serializable 肯定不能用。为什么有些公司要用 RC，或者说网上有些文章推

荐有 RC？

RC 和 RR 主要有几个区别：

1、 RR 的间隙锁会导致锁定范围的扩大。

2、 条件列未使用到索引，RR 锁表，RC 锁行。

3、 RC 的“半一致性”（semi-consistent）读可以增加 update 操作的并发性。

在 RC 中，一个 update 语句，如果读到一行已经加锁的记录，此时 InnoDB 返回记

录最近提交的版本，由 MySQL 上层判断此版本是否满足 update 的 where 条件。若满

足(需要更新)，则 MySQL 会重新发起一次读操作，此时会读取行的最新版本(并加锁)。

实际上，如果能够正确地使用锁（避免不使用索引去枷锁），只锁定需要的数据，

用默认的 RR 级别就可以了。

在我们使用锁的时候，有一个问题是需要注意和避免的，我们知道，排它锁有互斥

的特性。一个事务或者说一个线程持有锁的时候，会阻止其他的线程获取锁，这个时候

会造成阻塞等待，如果循环等待，会有可能造成死锁。

这个问题我们需要从几个方面来分析，一个是锁为什么不释放，第二个是被阻塞了

怎么办，第三个死锁是怎么发生的，怎么避免。

6 死锁

6.1 锁的释放与阻塞

回顾：锁什么时候释放？

事务结束（commit，rollback）；客户端连接断开。

如果一个事务一直未释放锁，其他事务会被阻塞多久？会不会永远等待下去？如果

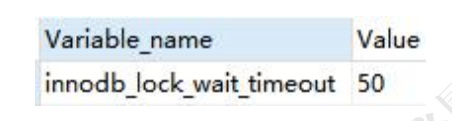
是，在并发访问比较高的情况下，如果大量事务因无法立即获得所需的锁而挂起，会占

用大量计算机资源，造成严重性能问题，甚至拖跨数据库。

[Err] 1205 - Lock wait timeout exceeded; try restarting transaction

MySQL 有一个参数来控制获取锁的等待时间，默认是 50 秒。

show VARIABLES like 'innodb\_lock\_wait\_timeout';



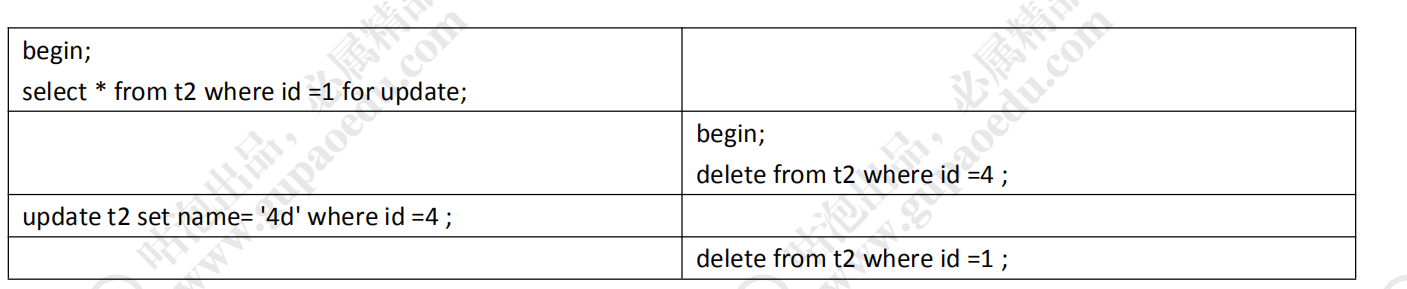
对于死锁，是无论等多久都不能获取到锁的，这种情况，也需要等待 50 秒钟吗？那

不是白白浪费了 50 秒钟的时间吗？

我们先来看一下什么时候会发生死锁。

6.2 死锁的发生和检测

死锁演示：



在第一个事务中，检测到了死锁，马上退出了，第二个事务获得了锁，不需要等待

50 秒：

[Err] 1213 - Deadlock found when trying to get lock; try restarting transaction

为什么可以直接检测到呢？是因为死锁的发生需要满足一定的条件，所以在发生死

锁时，InnoDB 一般都能通过算法（wait-for graph）自动检测到。

那么死锁需要满足什么条件？死锁的产生条件：

因为锁本身是互斥的，（1）同一时刻只能有一个事务持有这把锁，（2）其他的事

务需要在这个事务释放锁之后才能获取锁，而不可以强行剥夺，（3）当多个事务形成等

待环路的时候，即发生死锁。

举例：

理发店有两个总监。一个负责剪头的 Tony 总监，一个负责洗头的 Kelvin 总监。

Tony 不能同时给两个人剪头，这个就叫互斥。

Tony 在给别人在剪头的时候，你不能让他停下来帮你剪头，这个叫不能强行剥夺。

如果Tony的客户对Kelvin总监说：你不帮我洗头我怎么剪头？Kelvin的客户对Tony

总监说：你不帮我剪头我怎么洗头？这个就叫形成等待环路。

如果锁一直没有释放，就有可能造成大量阻塞或者发生死锁，造成系统吞吐量下降，

这时候就要查看是哪些事务持有了锁

6.3 查看锁信息（日志）

SHOW STATUS 命令中，包括了一些行锁的信息：

show status like 'innodb\_row\_lock\_%';



Innodb\_row\_lock\_current\_waits：当前正在等待锁定的数量；

Innodb\_row\_lock\_time ：从系统启动到现在锁定的总时间长度，单位 ms；

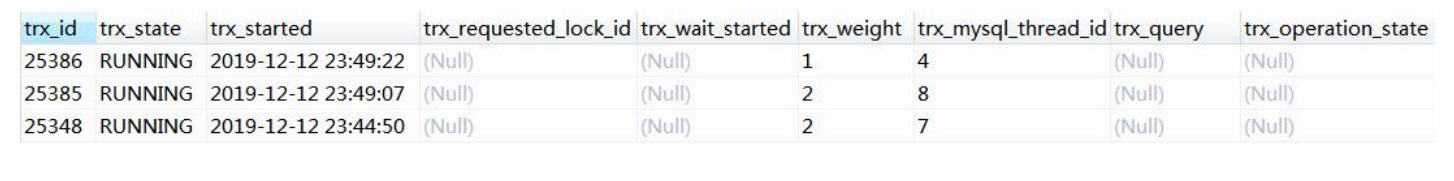
Innodb\_row\_lock\_time\_avg ：每次等待所花平均时间；

Innodb\_row\_lock\_time\_max：从系统启动到现在等待最长的一次所花的时间；

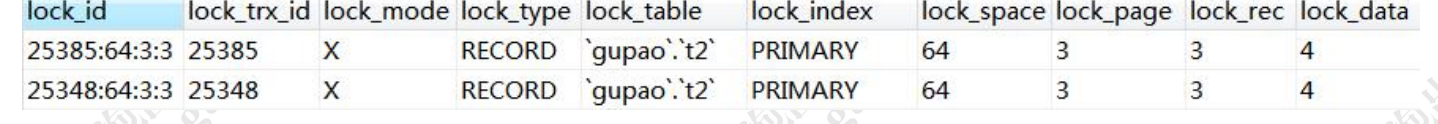
Innodb\_row\_lock\_waits ：从系统启动到现在总共等待的次数。

SHOW 命令是一个概要信息。InnoDB 还提供了三张表来分析事务与锁的情况：

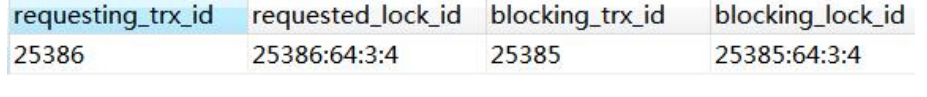
select \* from information\_schema.INNODB\_TRX; -- 当前运行的所有事务 ，还有具体的语句



select \* from information\_schema.INNODB\_LOCKS; -- 当前出现的锁



select \* from information\_schema.INNODB\_LOCK\_WAITS; -- 锁等待的对应关系



找出持有锁的事务之后呢？

如果一个事务长时间持有锁不释放，可以 kill 事务对应的线程 ID，也就是

INNODB\_TRX 表中的 trx\_mysql\_thread\_id，例如执行 kill 4，kill 7，kill 8。

当然，死锁的问题不能每次都靠 kill 线程来解决，这是治标不治本的行为。我们应该

尽量在应用端，也就是在编码的过程中避免。

有哪些可以避免死锁的方法呢？

6.4 死锁的避免

1、 在程序中，操作多张表时，尽量以相同的顺序来访问（避免形成等待环路）；

2、 批量操作单张表数据的时候，先对数据进行排序（避免形成等待环路）；

3、 申请足够级别的锁，如果要操作数据，就申请排它锁；

4、 尽量使用索引访问数据，避免没有 where 条件的操作，避免锁表；

5、 如果可以，大事务化成小事务；

6、 使用等值查询而不是范围查询查询数据，命中记录，避免间隙锁对并发的影响。

集中答疑帖：

https://gper.club/articles/7e7e7f7ff7g55gc8g6c