# 20 幻读是什么，幻读有什么问题？

CREATE TABLE `t` (

`id` int(11) NOT NULL,

`c` int(11) DEFAULT NULL,

`d` int(11) DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `c` (`c`)

) ENGINE=InnoDB;

insert into t values(0,0,0),(5,5,5),

(10,10,10),(15,15,15),(20,20,20),(25,25,25);

表除了主键id外，还有一个索引c,初始化语句插入6条数据

下面语句序列，是怎么加锁的，加的锁又是什么时候释放的？

begin;

select \* from t where d=5 for update;

commit;

这个语句会命中d=5 这一行，对应的主键是d=5,对应主键id=5,select 执行完，id=5这一行会加写锁，而且由于两阶段锁协议，这个写锁会在commit后释放。

由于字段d没有索引，所以会做全表扫描，其他被扫描的，但是不满足条件的会不会加锁呢？

**幻读是什么？**

现在分析，如果只在id=5这一行加锁，而其他的行不加锁，会怎么样？

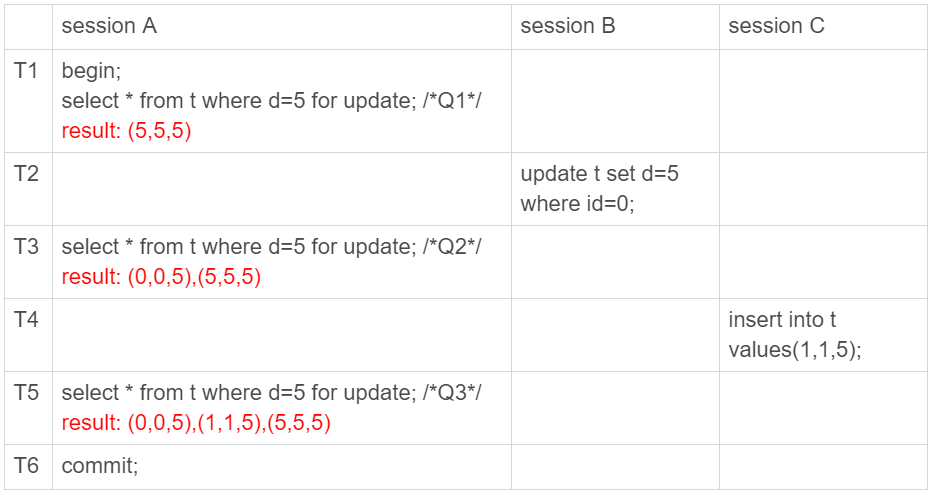


图 1 假设只在id=5这一行加锁

SessionA 执行了三次查询，分别是Q1,Q2,Q3,查询d=5的行，而且使用当前读，并且加上写锁，

分别返回什么结果

1. Q1只返回id=5这一行
2. 在T2时刻，sessionB把id=0这一行的d改成了5，因此T3时刻Q2查出来是id=0和id=5
3. 在T4时刻，sessionC 插入一行（1,1,5），因此T5时刻Q3查出来的id=0,id=1和id=5

Q3对到id=1这一行的现象称为“幻读”，幻读是指一个事务在前后两次查询同一个范围的时候，后一次查询到了前一次没有看到的行。

“幻读”说明

1. 在可重复读隔离级别下，普通的查询是快照读，是不会别的事务插入的数据的，因此，幻读在“当前读”下才会出现
2. 上面sessionB的修改结果，被sessionA之后的select语句用“当前读”看到，不能成为幻读。幻读仅专指“新插入的行”

从事务可见性分析，上面语句返回的结果没有问题。

因为这三个查询都加了for update,都是当前读，而当前读的规则，几时要能读到所有已经提交的记录最新值。并且sessionB和sessionC的两条语句，执行后就会提交，所以Q2和Q3就是应该看到这两个事务的操作效果，而且也看到了，着跟事务的可见性不矛盾

这里是有问题的

**幻读有什么问题**

**首先在语义上，**sessionA 在T1时刻就声明了，“我要把所有d=5的行锁住，不准别的事务进行读写操作”。实际上这个语义被破坏了。

往sessionB 和sessionC分别加入一条语句，会出现什么现象

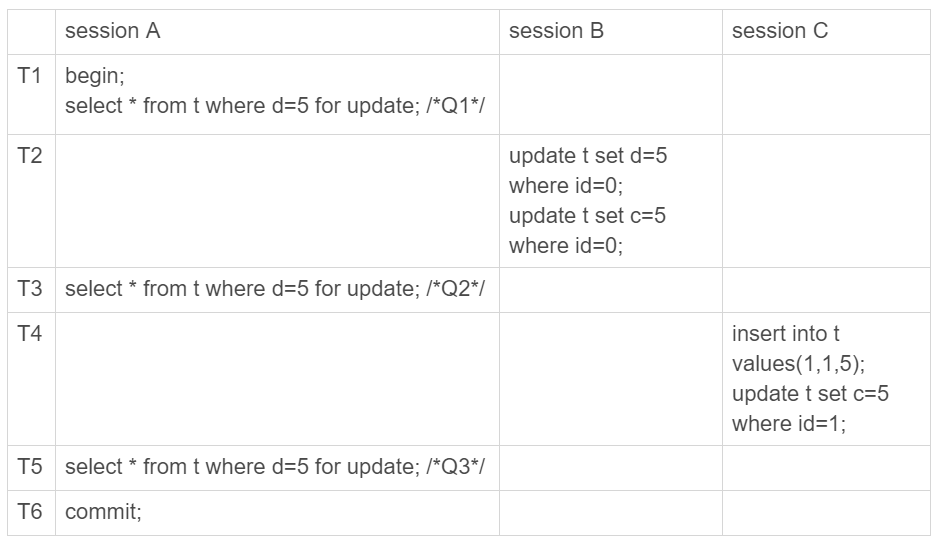


图2 假设只有id=5这一行加行锁-- 语义被破坏

Session B的第二条语句update t set c = 5 where id =0 ,语义是“我把id=0,d=5这一行的c值，改成了5”。

由于在T1时刻，session A还是只给id=5,这一行加了行锁，并没有给id=0这行加锁，因此sessionB在T2时刻，是可以执行这两条update语句的。这样就破坏了sessionA里Q1语句要锁住所有d=5的加锁的声明

SessionC 也是一样，对id=1这一行的修改，也破坏了Q1的加锁声明。

**其次，是数据一致性问题**

锁设计是为了保证数据的一致性，不只是数据库内部数据状态在此刻的一致性，还包含了数据和日志在逻辑上的一致性。

为了说明这个问题，给sessionA在T1时刻再加一个更新语句，即：update t set d=100 where d=5

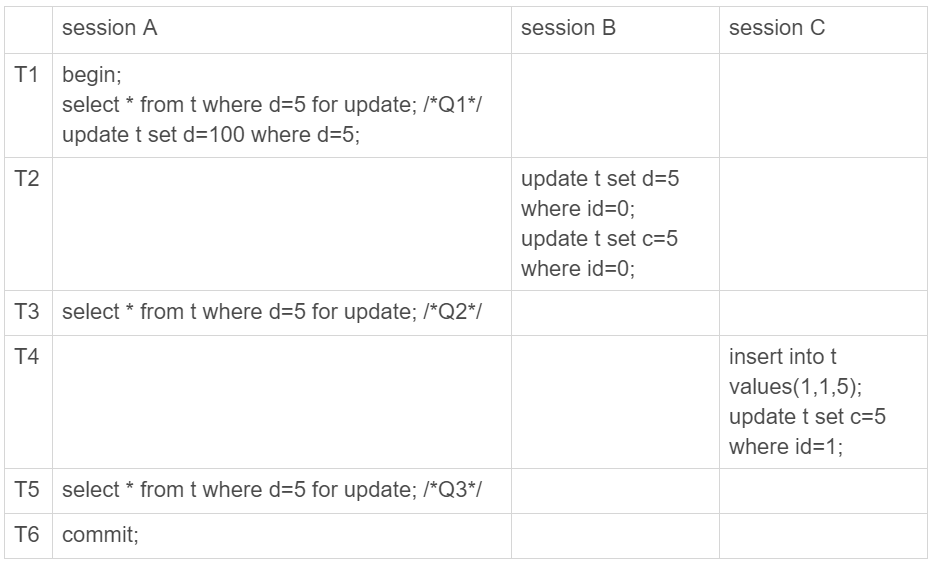


图3 假设只在id=5这一行加行锁 -- 数据一致性问题

Update的加锁语义和select...for update是一致的，所以此时加上这条update语句和很合理。sessionA声明说“要给d=5的语句加上锁”，就是为了要更新数据，新加的这条update语句就是把它认为加上了锁的这一行的d改成100。

分析图3执行完成后，数据库是什么结果

1.经过T1时刻，id=5这一行变成（5,5,100）当然这个最终结果是在T6时刻正式提交

2.经过T2时刻，id=0这一行变成了（0,5,5）

3.经过T4时刻，表里面多了一行（1,5,5）

4其他行跟这个执行序列无关，保持不变

这样看，这些数据也没啥问题，但是看看binlog里面的内容

1. T2时刻，sessionB事务提交，写入两条语句
2. T4时刻，sessionC 事务提交，写入两条语句
3. T6时刻，session A事务提交，写入了update t set d=100 where d=5这条语句

同一放一起的话，

update t set d=5 where id=0; /\*(0,0,5)\*/

update t set c=5 where id=0; /\*(0,5,5)\*/

insert into t values(1,1,5); /\*(1,1,5)\*/

update t set c=5 where id=1; /\*(1,5,5)\*/

update t set d=100 where d=5;/\* 所有 d=5 的行，d 改成 100\*/

问题出现了，这个语句序列，不论拿到备库执行，还是以后用binlog来克隆一个库，这三行的结果都变成了（0,5,100）（1,5,100）和（5,5,100）。

也就是说id=0和id=1这两行，发生数据不一致，问题很严重

到这里，**这个数据不一致到底是怎么引入的？**

**假设**“select \* from t where d=5 for update这条语句只给d=5这一行，也就是id=5的这一行加锁”导致的。

所以认为，上面的设定不合理，需要改

把扫描过程中碰到的行，也都加上写锁，再来看看执行效果

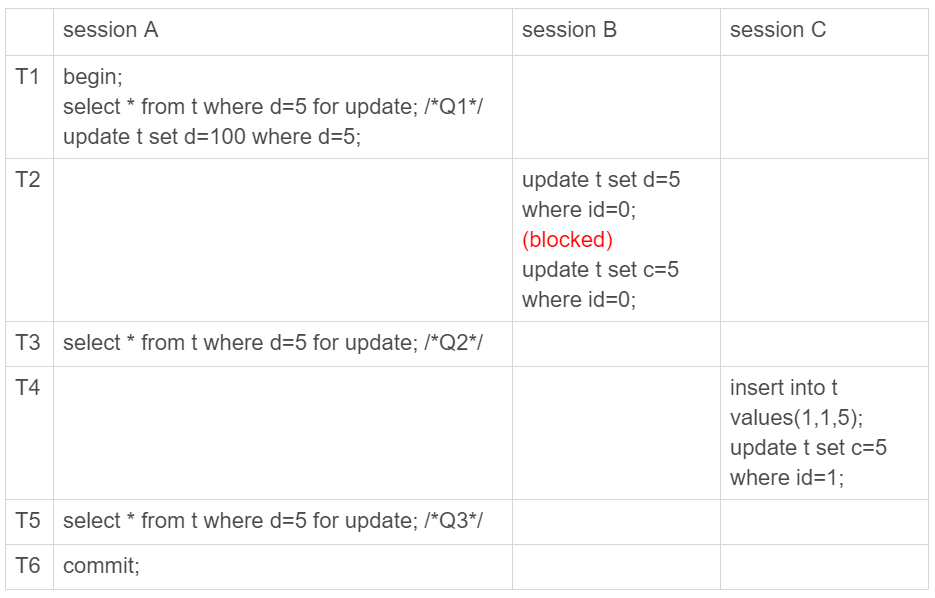


图 4 假设扫描到的行都被加上了行锁

由于sessionA把所有的行都加上了行锁，所以sessionB在执行第一个update语句的时候被锁住了。需要等到T6时刻sessionA提交以后，sessionB才能继续执行

这样对于id=0这一行，在数据库里的最终的结果还是（0，5，5），在binlog里面，

insert into t values(1,1,5); /\*(1,1,5)\*/

update t set c=5 where id=1; /\*(1,5,5)\*/

update t set d=100 where d=5;/\* 所有 d=5 的行，d 改成 100\*/

update t set d=5 where id=0; /\*(0,0,5)\*/

update t set c=5 where id=0; /\*(0,5,5)\*/

按照日志执行，id=0这一行的最终结果也是（0,5,5）。所以id=0这一行的问题解决了

Id=1这一行，在数据库里面的结果是（1,5,5），而binlog的执行结果是（1,5,100）也就是幻读的问题依然存在，为什么吧所有的记录都加上了锁，还是阻止不了id=1这一行的插入和更新呢？

在T3时刻，给所有的行加锁的时候，id=1这一行还不存在，不存在的也就加不上锁

**也就是说，即使把所有的记录都加上锁，还是阻止不了新插入的记录，**这也是为什么“幻读”会被单独拿出来解决的原因。

**如何解决幻读？**

产生幻读的原因是，行锁只能锁住行，但是新插入记录这个动作，要更新的是记录之间的“间隙”。因此，为了解决幻读问题，InnoDB只好引入新的锁，也就是间隙锁（GapLock）

间隙锁，锁的就是两个值之间的空隙，不如文章开头的表t，初始化插入了6个数据，产生了7个间隙

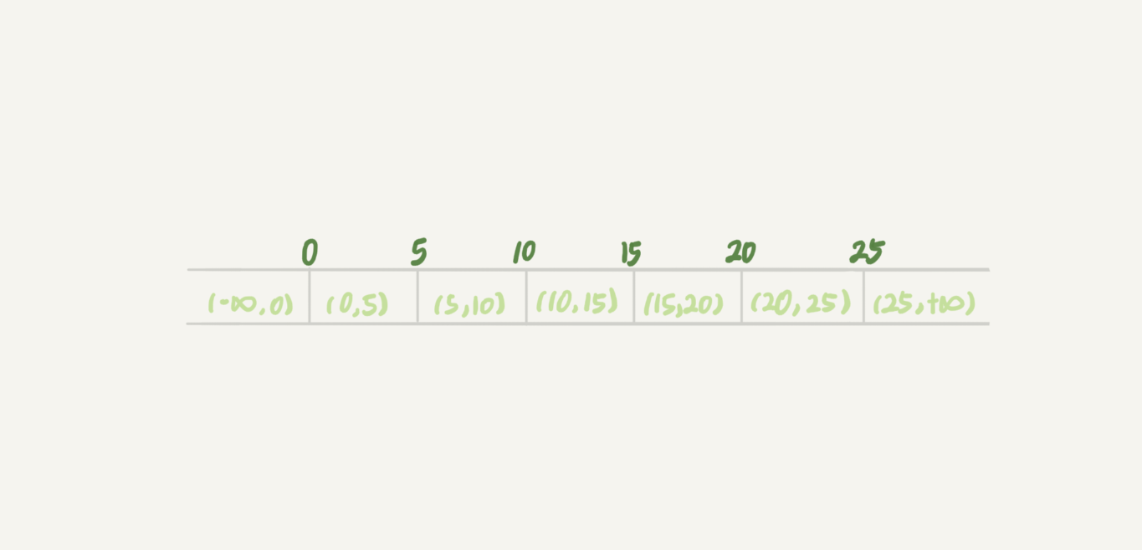


图 5 表t 主键索引上的行锁和间隙锁

当执行 select \* from t where d=5 for update的时候，就不止是给数据库中已有的6个记录加上了行锁，还同时加了7个间隙锁，这样确实保证了无法插入新的记录

在一行行扫描的过程中，不仅将给行加上了行锁，还给行两边的空隙，也加上了间隙锁

数据行时可以加上锁的实体，数据行之间的间隙，也是可以加上锁的实体，但是间隙锁和之前的锁不太一样

比如行锁，分为读锁和写锁，下图是两种行锁的冲突关系

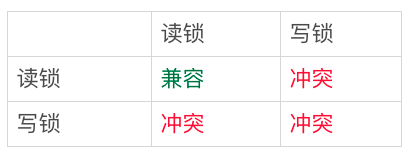


图 6 两种行锁之间的冲突关系

跟行锁有冲突的关系是“另外一个行锁”

但是间隙锁不一样，**跟间隙锁存在冲突关系的，是往这个间隙中间插入一个记录的操作，**间隙锁都不存在冲突关系

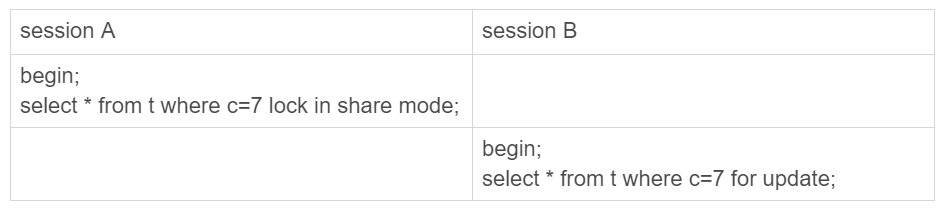


图 7 间隙锁之间不互锁

SessionB并不会被堵住。因为表t里并没有c=7这个记录，因此sessionA加的是间隙锁（5,10）

而sessionB也是在这个间隙加的间隙锁，他们有共同的目标，即：保护这个间隙，不允许插入值，它们之间不冲突。

间隙锁和行锁合成next-key lock，每个next-key lock是前开后闭区间，表t初始化后，如果用select \* from t for update 要把整个表所有记录锁起来，就形成了7个next-key lock,分别(-∞,0]、(0,5]、(5,10]、(10,15]、(15,20]、(20, 25]、(25, +supremum]。

**间隙锁和next-key lock的引入，帮助解决了幻读的问题，但同时也带来了一些“困扰”**

有个问题：任意锁住一行，如果这一行不存在的话就插入，如果存在这一行就更新数据，

begin;

select \* from t where id=N for update;

/\* 如果行不存在 \*/

insert into t values(N,N,N);

/\* 如果行存在 \*/

update t set d=N set id=N;

Commit;

这个不是insert... on duplicate key update就能解决吗？但其实有多个唯一键的时候，这个方法是不满足需求的，为什么，以后说

讨论这个逻辑，一点并发，就会碰到死锁，这个逻辑每次操作前用for update锁起来，已经是最严格的模式，怎会死锁呢？

设置两个session ,并假设N=9

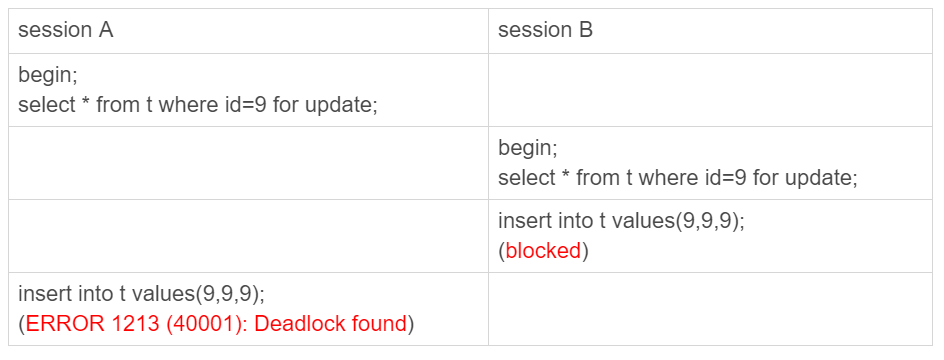


图8 间隙锁导致的死锁

1.sessionA 执行select... for update语句，由于id=9这一行不存在，因此会加上间隙锁（5,10）

2.sessionB 执行 select...for update 语句，同样会加上间隙锁（5,10），间隙锁之间不会冲突，因此这个语句可以执行成功。

3.sessionB 视图插入一行（9,9,9），被sessionA间隙锁挡住了，只好进入等待

4.sessionA 视图插入一样（9,9,9），被sessionB 间隙锁挡住了。

至此，两个session进入相互等待，形成死锁，InnoDB的死锁检测马上就这对死锁关系，让sessionA的insert语句报错返回

**间隙锁的引入，可能会导致同样的语句锁住更大的范围，这其实是影响了并发度的，**其实这还是一个简单的例子。

间隙锁是在可重复读隔离级别下才会生效的，所以把隔离级别设置为读提交的话，就没有加息锁了，但同时，要解决可能出现的binlog数据和日志不一致的问题，需要把binlog格式设置为row,也是不少公司使用的配置组合。

思考题

为什么sessionB和sessionC的insert语句都会进入锁等待状态

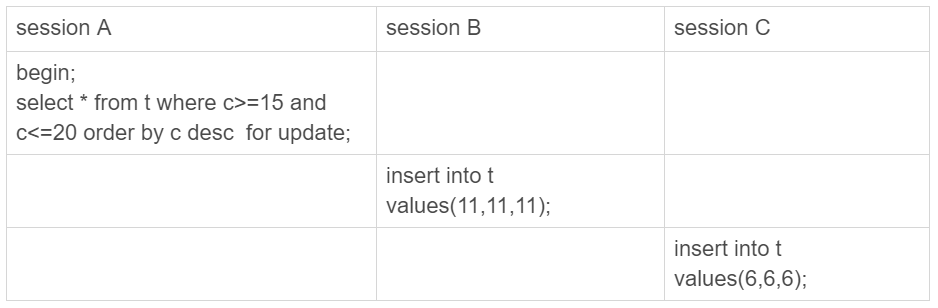


图9 事务进入锁等待状态