40 | insert语句的锁为什么这么多

**Insert...select 语句**

初始化表

CREATE TABLE `t` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`c` int(11) DEFAULT NULL,

`d` int(11) DEFAULT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `c` (`c`)

) ENGINE=InnoDB;

insert into t values(null, 1,1);

insert into t values(null, 2,2);

insert into t values(null, 3,3);

insert into t values(null, 4,4);

create table t2 like t

在可重复读隔离级下，binlog\_format=statement是执行

insert into t2(c,d) select c,d from t;

这个语句时，需要对表t的所有行和间隙加锁？

这个问题需要考虑的还是日志和数据的一致性，看看下面的执行序列

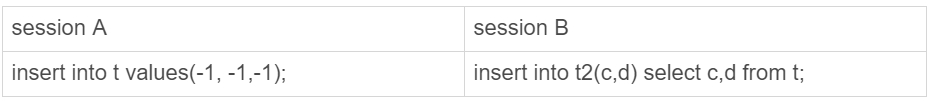


图 1 并发insert 场景

实际的执行效果是，如果sessionB 先执行，由于这个表语句对表t主键索引加了(-∞，1]这个next-key lock，会在执行完成后，才允许sessionA的insert语句执行。

如果没有锁的话，就可能出现sessionB的insert语句先执行，但是后写binlog的情况，于是，binlog\_format=statement的情况下，binlog里面就记录了这样的语句

insert into t values(-1,-1,-1);

insert into t2(c,d) select c,d from t;

**Insert循环写入**

执行insert...select时，对目标表也不是锁全表，而只是锁住需要访问的资源

如果有这么一个需求：要往表t2中插入一行数据，这一行的c的字是表t中c值的最大值加1

这么写SQL

insert into t2(c,d) (select c+1, d from t force index(c) order by c desc limit 1);

这个语句的加锁范围，就是表t索引c上的(3,4]和(4,supremum]这两个next-key lock,以及主键索引上id=4这一行。

执行流程比较简单，从表t中按照索引c倒序，扫描第一行，拿到结果写入到表t2中，因此整条语句的扫描行数是1

这个语句执行的慢查询日志（slow log），如图所示

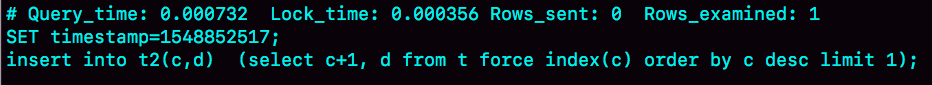


图2 慢查询日志 -- 将数据出入表t2

通过慢查询日志，看到Rows\_examined=1,正好验证了执行这条语句的扫描行数为1

那么，如果要把这样的一行数据插入到表t中

insert into t(c,d) (select c+1, d from t force index(c) order by c desc limit 1);

语句的执行流程是怎么样的？扫描行数有事多少呢？

此时，在看慢查询日志就户发现不对了

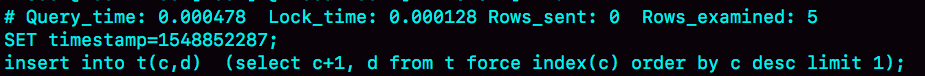


图 3 慢查询日志 -- 将数据插入表t

此时 Rows\_examined的值是5.

如图4 就是语句的explain结果

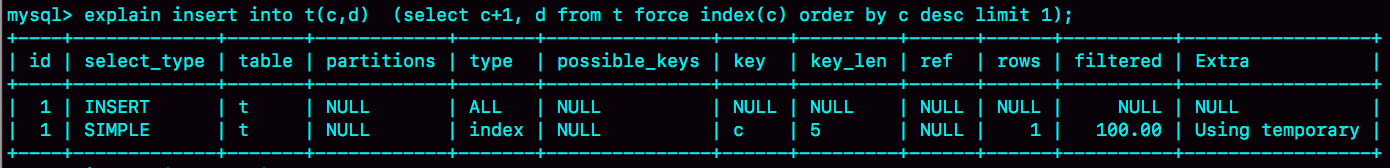


图4 explain 结果

从Extra 字段可以看到“Using temporary”字样，表示这个语句用到了临时表，执行过程中，需要把表t的内容读出来，写入临时表。

果读出来（扫描1行），写入临时表，然后再从临时表读出来（扫描1行），写回表t中，那么，这个语句的扫描行数就应该是2，而不是5.

所以这个猜测不对，实际上Explain结果里rows=1，是因为受到了limit=1的影响

从另一个角度考虑，可以看看InnoDB扫描了多少行，如图5所示，是在执行这个语句前后查看innodb\_rows\_read的结果

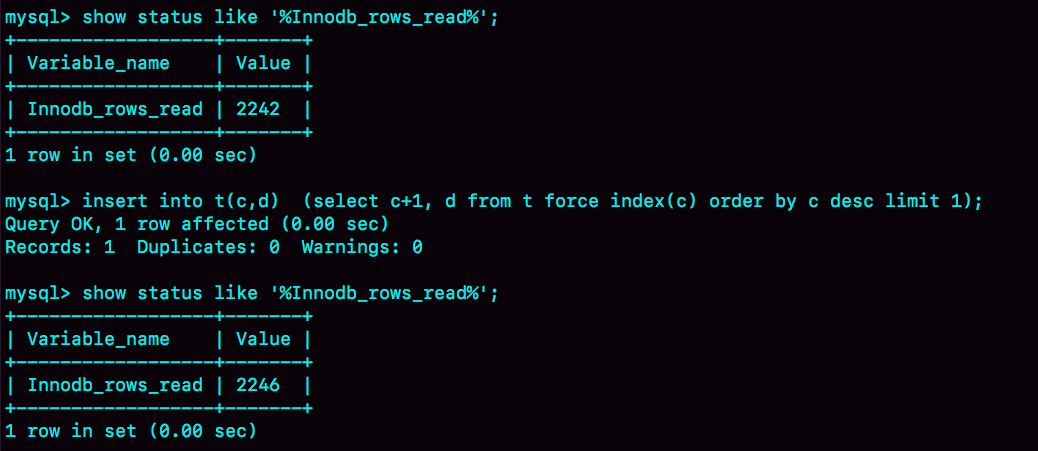


图 5 查看Innodb\_rows\_read变化

这个语句执行前后,innodb\_rows\_read的值增加了4，因为默认临时表是使用Memory引擎的，所以这个4行查的都是表t,也就是对表t做了全表扫描。

整个执行流程

1. 创建临时表，表里有两个字段c和d.
2. 按照索引c扫描表t，依次取c=4,3,2,1,然后回表，读到c和d的值写入临时表，此时，Rows\_examined=4.
3. 由于语义里面有limit 1,所以只取了临时表的第一行，再插入到表t中，此时，Rows\_examined的值加1，变成了5.

整个语句会导致在表t上左全表扫描，并且 给索引c上的所有间隙都加上共享的next-key lock.所以，整个语句中期间，其他事务不能在这个表上插入数据

至于这个语句的执行为什么需要临时表，原因是这类一边遍历数据，一边更新数据的情况。、如果读出来的数据是直接写回原表，就可能在遍历的过程中，读到刚刚插入的记录，新纪录的插入如果参与计算逻辑，就跟语义不符。

由于实现上这个语句没有在子查询中就直接用limit=1，从而导致了这个语句的执行需要遍历整个表t。他的优化方法简单，就是用前面介绍过的方法，先insert into 到临时表temp\_t，这样就只需要扫描一行，然后再从表temp\_t里面取这行数据插入表t1.

由于这个语句涉及的数据量很小，可以考虑使用内存临时表做这个优化

create temporary table temp\_t(c int,d int) engine=memory;

insert into temp\_t (select c+1, d from t force index(c) order by c desc limit 1);

insert into t select \* from temp\_t;

drop table temp\_t;

**Insert 唯一键冲突**

前面的两个例子使用insert...select的情况，下面的例子是常见的insert语句出现唯一键冲突的情况。

对于有唯一键的表，插入数据时出现唯一键冲突也是常见的情况了，唯一键冲突例子



图 6 唯一键冲突加锁

在可重复读隔离级别下，sessionB要执行的insert语句进入锁等待状态

SessionA 执行的insert语句，发生唯一键冲突的时候，并不只是简单的报错返回，还在冲突的索引上加了锁，一个next-key lock就是由它右边界的值定义的，此时， sessionA持有索引c上的(5,10]共享next-key lock(读锁)

至于为什么要加这个锁，从作用上看是，可以避免这一行被别的事务删除。

一个经典死锁案例



图 7 唯一键冲突-- 死锁

在session A执行rollback语句回滚时，sessionC几乎同时发现死锁并返回

这个死锁产生的逻辑是这样的

1. 在T1时刻，启动sessionA,并执行insert语句，此时在索引c的c=5上加了记录锁，这个索引是唯一索引，因此退化为记录锁（回顾第21篇）
2. 在T2时刻，sessionB要执行相同的insert语句，发现唯一键冲突，加上读锁，同样的，sessionC也在索引c上，c=5这个记录，加了读锁
3. T3时刻，sessionA回滚，此时，sessionB和sessionC都试图继续中插入操作，都要加上写锁，两个session都要等待对方的行锁，所以出现死锁。

Insert into ... on duplicate key update

上面的例子是主键冲突后直接报错，如果改写成

insert into t values(11,10,10) on duplicate key update d=100;

就会给索引c上(5,10]加上一个排他的next-key lock（写锁）

**Insert into ...on duplicate key update这个语义的逻辑是，插入一行数据，如果碰到唯一键约束，就执行后面的更新语句**

如果有多个列违反唯一性约束，就会按照索引的顺序，修改跟第一个索引冲突的行。

现有表t里已经有（1，1，1）和（2，2，2）这两行，看看这个语句的执行流程

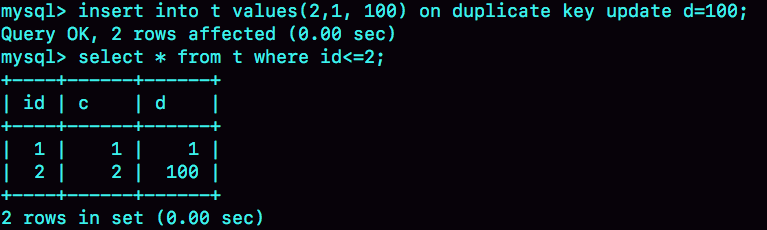


图 9 两个唯一键同时冲突

主键是先判断的，MySQL认为这个语句跟id=2这一行冲突，所以修改的是id=2的行，

需要注意的是，执行这条语句的affected rows返回的是2，很容易误解。实际上，真正更新的只有一行，只是代码上，insert和update都认为自己成功了，update计数加1，insert计数也加了1.

**思考题**

平时两个表指之间拷贝数据用什么方法，有什么注意事项吗？在应用场景里，这个方法，相较于其他方法的优势是什么？