mysql如何防止幻读

原创 我们都爱松松吧 已于 2022-04-28 22:11:08 修改 1848 收藏 7 版权

分类专栏： MySQL 文章标签： mysql 数据库 java

MySQL 专栏收录该内容

前言

以下内容是作者在网上搜集和自己总结而来。

一、基本概念

MySQL 事务都是指在 InnoDB 引擎下，MyISAM 引擎是不支持事务的。

事务具有 原子性 （Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Durability）四个特性，简称 ACID，缺一不可。今天要说的就是隔离性。

1.1 脏读

脏读指的是读到了其他事务未提交的数据，未提交意味着这些数据可能会 回滚 ，也就是可能最终不会存到数据库中，也就是不存在的数据。读到了并一定最终存在的数据，这就是脏读。

1.2 可重复读

可重复读指的是在一个事务内，最开始读到的数据和事务结束前的任意时刻读到的同一批数据都是一致的。通常针对数据更新（UPDATE）操作。

1.3 不可重复读

对比可重复读，不可重复读指的是在同一事务内，不同的时刻读到的同一批数据可能是不一样的，可能会受到其他事务的影响，比如其他事务改了这批数据并提交了。通常针对数据更新（UPDATE）操作。

1.4 幻读

幻读是针对数据插入（INSERT）操作来说的。假设事务A对某些行的内容作了更改，但是还未提交，此时事务B插入了与事务A更改前的记录相同的记录行，并且在事务A提交之前先提交了，而这时，在事务A中查询，会发现好像刚刚的更改对于某些数据未起作用，但其实是事务B刚插入进来的，让用户感觉很魔幻，感觉出现了幻觉，这就叫幻读。

二、事务隔离级别

SQL 标准定义了四种隔离级别，MySQL 全都支持。这四种隔离级别分别是：

2.1 读未提交（READ UNCOMMITTED）

最低的事务隔离级别，一个事务还没提交时，它做的变更就能被别的事务看到；

2.2 读提交 （READ COMMITTED）

保证一个事物提交后才能被另外一个事务读取。另外一个事务不能读取该事物未提交的数据，可避免脏读的发生，但是可能会造成不可重复读；

2.3 可重复读 （REPEATABLE READ）

多次读取同一范围的数据会返回第一次查询的快照，即使其他事务对该数据做了更新修改。事务在执行期间看到的数据前后必须是一致的，但如果这个事务在读取某个范围内的记录时，其他事务又在该范围内插入了新的记录，当之前的事务再次读取该范围的记录时，会产生幻行，这就是幻读；

2.4 串行化 （SERIALIZABLE）

花费最高代价但最可靠的事务隔离级别。

“写”会加“写锁”，“读”会加“读锁”。当出现读写锁冲突的时候，后访问的事务必须等前一个事务执行完成，才能继续执行

只有串行化的隔离级别解决了全部这 3 个问题，其他的 3 个隔离级别都有缺陷。
从上往下，隔离强度逐渐增强，性能逐渐变差。采用哪种隔离级别要根据系统需求权衡决定，其中，可重复读是 MySQL 的默认级别。

事务隔离其实就是为了解决上面提到的脏读、不可重复读、幻读这几个问题，下面展示了 4 种隔离级别对这三个问题的解决程度。

隔离级别	脏读	不可重复读	幻读
读未提交	可能	可能	可能
读提交	不可能	可能	可能
可重复读	不可能	不可能	可能
串行化	不可能	不可能	不可能

只有串行化的隔离级别解决了全部这 3 个问题，其他的 3 个隔离级别都有缺陷。

三、快照读–MVCC

3.1 什么是MVCC

MVCC，全称Multi-Version Concurrency Control，即多版本并发控制。MVCC是一种并发控制的方法，一般在数据库管理系统中，实现对数据库的并发访问，在编程语言中实现事务内存。

MVCC在MySQL InnoDB中的实现主要是为了提高数据库并发性能，用更好的方式去处理读-写冲突，做到即使有读写冲突时，也能做到不加锁，非阻塞并发读

3.2 什么是快照读

像不加锁的select操作就是快照读，即不加锁的非阻塞读；快照读的前提是隔离级别不是串行级别，串行级别下的快照读会退化成当前读；之所以出现快照读的情况，是基于提高并发性能的考虑，快照读的实现是基于多版本并发控制，即MVCC,可以认为MVCC是行锁的一个变种，但它在很多情况下，避免了加锁操作，降低了开销；既然是基于多版本，即快照读可能读到的并不一定是数据的最新版本，而有可能是之前的历史版本

3.3 快照读的实现方式

MVCC中的核心知识点

（1）事务版本号

每次事务开启前都会从数据库获得一个自增长的事务ID，可以从事务ID判断事务的执行先后顺序。

（2）表的隐藏列

DB_TRX_ID: 记录操作该数据事务的事务ID；

DB_ROLL_PTR：指向上一个版本数据在undo log 里的位置指针；

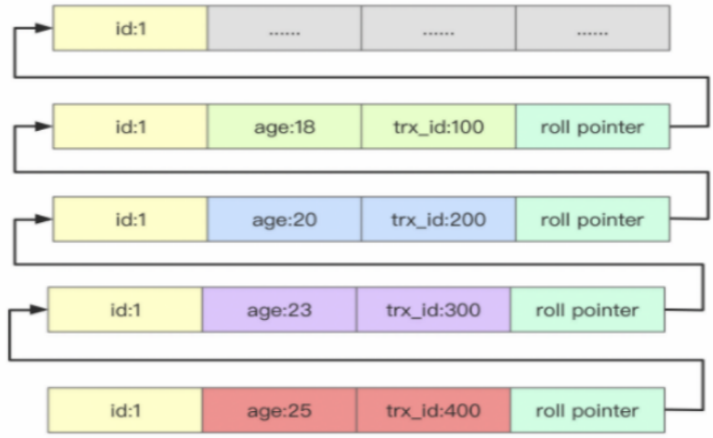
DB_ROW_ID: 隐藏ID ， 当创建表没有合适的索引作为聚集索引时，会用该隐藏ID创建聚集索引;

（3）undo log

Undo log 主要用于记录数据被修改之前的日志，在表信息修改之前先会把数据拷贝到undo log 里，当事务进行回滚时可以通过undo log 里的日志进行数据还原。

undo log是一种逻辑日志，当一个事务对记录做了变更操作就会产生undo log，也就是说undo log记录了记录变更的逻辑过程。当一个事务要更新一行记录时，会把当前记录当做历史快照保存下来，多个历史快照会用两个隐藏字段trx_id和roll_pointer串起来（关于隐藏字段，这里不用考虑隐式主键id:DB_ROW_ID），形成一个历史版本链。可以用于MVCC和事务回滚。

比如多个事务对id为1的数据做了更新，会形成如下图这种历史版本链：



CSDN @我们都爱松松吧

（4）read view

在innodb 中每个SQL语句执行前都会得到一个read_view。副本主要保存了当前数据库系统中正处于活跃（没有commit）的事务的ID号，其实简单的说这个副本中保存的是系统中当前不应该被本事务看到的其他事务id列表。

min_trx_id：read view生成时，活跃事务id列表中的最小id

max_trx_id：read view生成时，数据库即将分配的事务id，也就是当前已创建最大事务id+1

一个事务在对一行数据做读取操作的时候，会从undo log历史版本链中从最新版本开始往前比对，通过一系列的规则，根据快照版本中的trx_id字段和read view来确定该版本对于当前事务是否可见，如果当前比对版本不可见，那么就通过roll_pointer找到上一个版本进行比对，直到找到可见版本或找不到任何一个可见版本。这些规则定义如下：

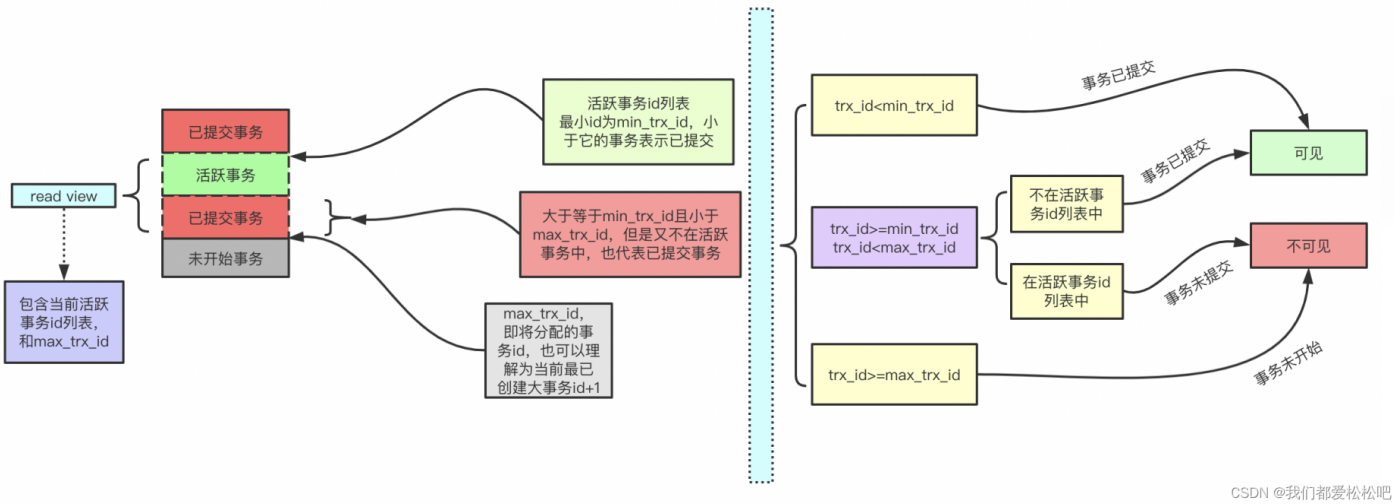
1.如果 trx_id < min_trx_id，则说明该版本对于当前事务(read view)来说，是已提交事务生成的，那么对于当前事务可见。

2.如果trx_id >= max_trx_id：则说明说明该数据是在当前read view 创建之后才产生的，所以数据不予显示。3.如果min_trx_id =< trx_id < max_trx_id：这种情况就说明这个数据有可能是在当前事务开始的时候还没有提交的。所以这时候我们需要把数据的事务ID与当前read view 中的活跃事务集合 trx_ids 匹配:

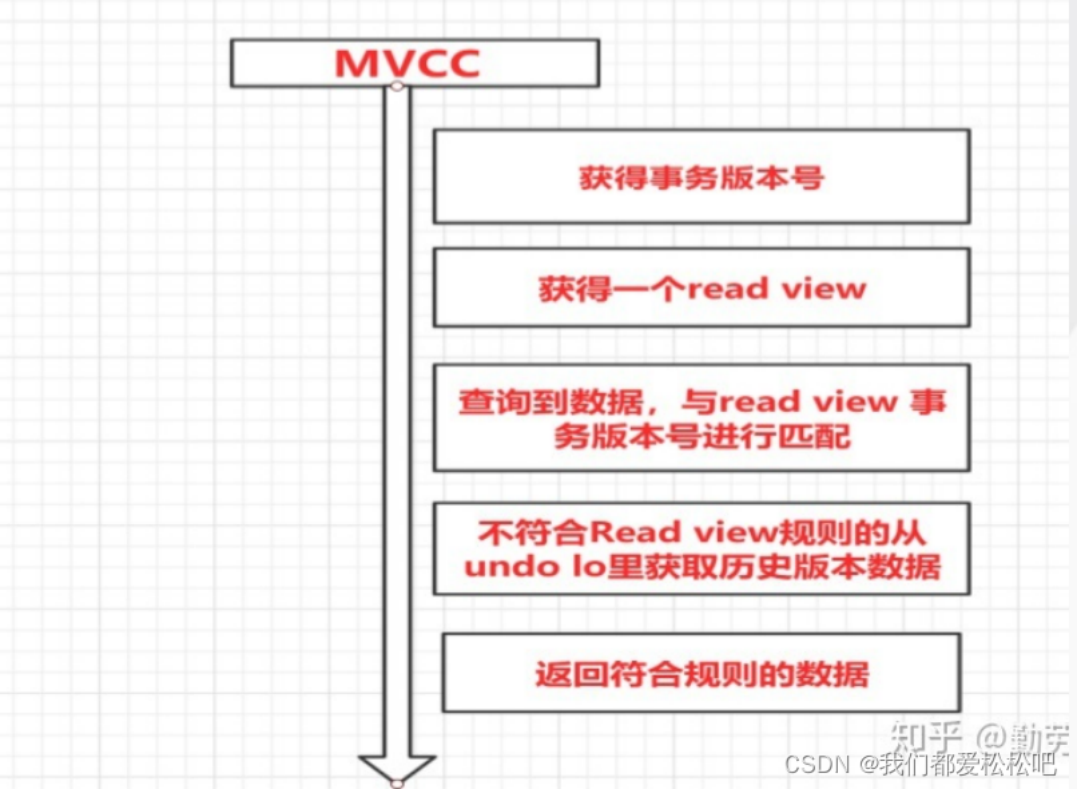
情况1: 如果事务ID不存在于trx_ids 集合（则说明read view产生的时候事务已经commit了），这种情况数据则可以显示。

情况2： 如果事务ID存在trx_ids则说明read view产生的时候数据还没有提交，但是如果数据的事务ID等于creator_trx_id ，那么说明这个数据就是当前事务自己生成的，自己生成的数据自己当然能看见，所以这种情况下此数据也是可以显示的。

情况3： 如果事务ID既存在trx_ids而且又不等于creator_trx_id那就说明read view产生的时候数据还没有提交，又不是自己生成的，所以这种情况下此数据不能显示。



CSDN @我们都爱松松吧



思考下面一个问题:

在RC（读已提交）和RR（可重复度）级别下，MVCC都会生效，那么为什么RC不可以解决幻读，而RR可以解决幻读?

```
原因：  两种隔离界别下的核心处理逻辑就是判断所有版本中哪个版本是当前事务可见的处理。针对这个问题InnoDB在设计上增加了ReadView的设计，

以上内容是对于 RR 级别来说，而对于 RC 级别，其实整个过程几乎一样，唯一不同的是生成 ReadView 的时机，RR 级别只在事务开始时生成一次，
```

四、当前读–锁

4.1 锁介绍

当前读是读取的数据库最新的数据，当前读和快照读不同，因为要读取最新的数据而且要保证事务的隔离性，所以当前读是需要对数据进行加锁的，当前读的实现方式就是Next key lock临键锁。

临键锁(Next-key Locks)
临键锁，是记录锁与间隙锁的组合，它的封锁范围，既包含索引记录，又包含索引区间

记录锁(Record Locks)
记录锁是 封锁记录，记录锁也叫行锁，例如：
SELECT * FROM student WHERE id =1 FOR UPDATE;
它会在 id=1 的记录上加上记录锁，以阻止其他事务插入，更新，删除 id=1 这一行。

间隙锁(Gap Locks)
间隙锁是封锁索引记录中的间隔，或者第一条索引记录之前的范围，又或者最后一条索引记录之后的范围.

4.2 产生间隙锁条件

产生间隙锁的条件（RR事务隔离级别下；）：

1.使用普通索引锁定；
2.使用多列唯一索引；
3.使用唯一索引锁定多行记录。

打开间隙锁设置

首先查看 innodb_locks_unsafe_for_binlog 是否禁用：
show variables like 'innodb_locks_unsafe_for_binlog';

Variable_name	Value
innodb_locks_unsafe_for_binlog	OFF

默认值为OFF，即启用间隙锁。因为此参数是只读模式，如果想要禁用间隙锁，需要修改 my.cnf（windows是my.ini） 重新启动才行

4.3 加锁规则

加锁规则有以下特性，我们会在后面的案例中逐一解释：

1.加锁的基本单位是（next-key lock）,他是前开后闭原则
2.插叙过程中访问的对象会增加锁
3.索引上的等值查询--给唯一索引加锁的时候，next-key lock升级为行锁
4.索引上的等值查询--向右遍历时最后一个值不满足查询需求时，next-key lock 退化为间隙锁
5.唯一索引上的范围查询会访问到不满足条件的第一个值为止

4.4 案例数据

案例数据

id(主键)	c（普通索引）	d（无索引）
5	5	5
10	10	10
15	15	15
20	20	20
25	25	25

以上数据为了解决幻读问题，更新的时候不只是对上述的五条数据增加行锁，还对于中间的取值范围增加了6间隙锁，（-∞，5]（5，10]（10，15]（15，20]（20，25]（25，+supernum]（其中supernum是数据库维护的最大的值。为了保证间隙锁都是左开右闭原则。）

案例二： 间隙锁死锁问题

步骤	事务A	事务B
1	begin; select * from t where id = 9 for update;	-
2	-	begin; select * from t where id = 6 for update;
3	-	insert into user value(7,7,7) *blocked*
4	insert into user value(7,7,7) *blocked*	-

不同于写锁相互之间是互斥的原则，间隙锁之间不是互斥的，如果一个事务A获取到了（5,10]之间的间隙锁，另一个事务B也可以获取到（5,10]之间的间隙锁。这时就可能会发生死锁问题，如下案例。

事务A获取到（5,10]之间的间隙锁不允许其他的DDL操作，在事务提交，间隙锁释放之前，事务B也获取到了间隙锁（5,10]，这时两个事务就处于死锁状态

案例三： 等值查询—唯一索引

步骤	事务A	事务B	事务C
1	begin; update u set d= d+ 1 where id = 7;	-	-
2	-	insert into u (8,8,8); *blocked*	-
4	-	-	update set d = d+ 1 where id = 10

1.加锁的范围是（5,10]的范围锁

2.由于数据是等值查询，并且表中最后数据id = 10 不满足id= 7的查询要求，故id=10 的行级锁退化为间隙锁，（5,10）

3.所以事务B中id=8会被锁住，而id=10的时候不会被锁住

4.5 临键锁

临键锁(Next-key Locks)
临键锁，是记录锁与间隙锁的组合，它的封锁范围，既包含索引记录，又包含索引区间。

注：临键锁的主要目的，也是为了避免幻读(Phantom Read)。如果把事务的隔离级别降级为RC，临键锁则也会失效。

总结

总结：

1.记录锁、间隙锁、临键锁，都属于排它锁；
2.记录锁就是锁住一行记录；
3.间隙锁只有在事务隔离级别 RR 中才会产生；
4.唯一索引只有锁住多条记录或者一条不存在的记录的时候，才会产生间隙锁，指定给某条存在的记录加锁的时候，只会加记录锁，不会产生间隙锁；
5.普通索引不管是锁住单条，还是多条记录，都会产生间隙锁；
6.间隙锁会封锁该条记录相邻两个键之间的空白区域，防止其它事务在这个区域内插入、修改、删除数据，这是为了防止出现 幻读 现象；
7.普通索引的间隙，优先以普通索引排序，然后再根据主键索引排序（多普通索引情况还未研究）；
8.事务级别是RC（读已提交）级别的话，间隙锁将会失效。

思考

MVCC真的解决了幻读问题吗?