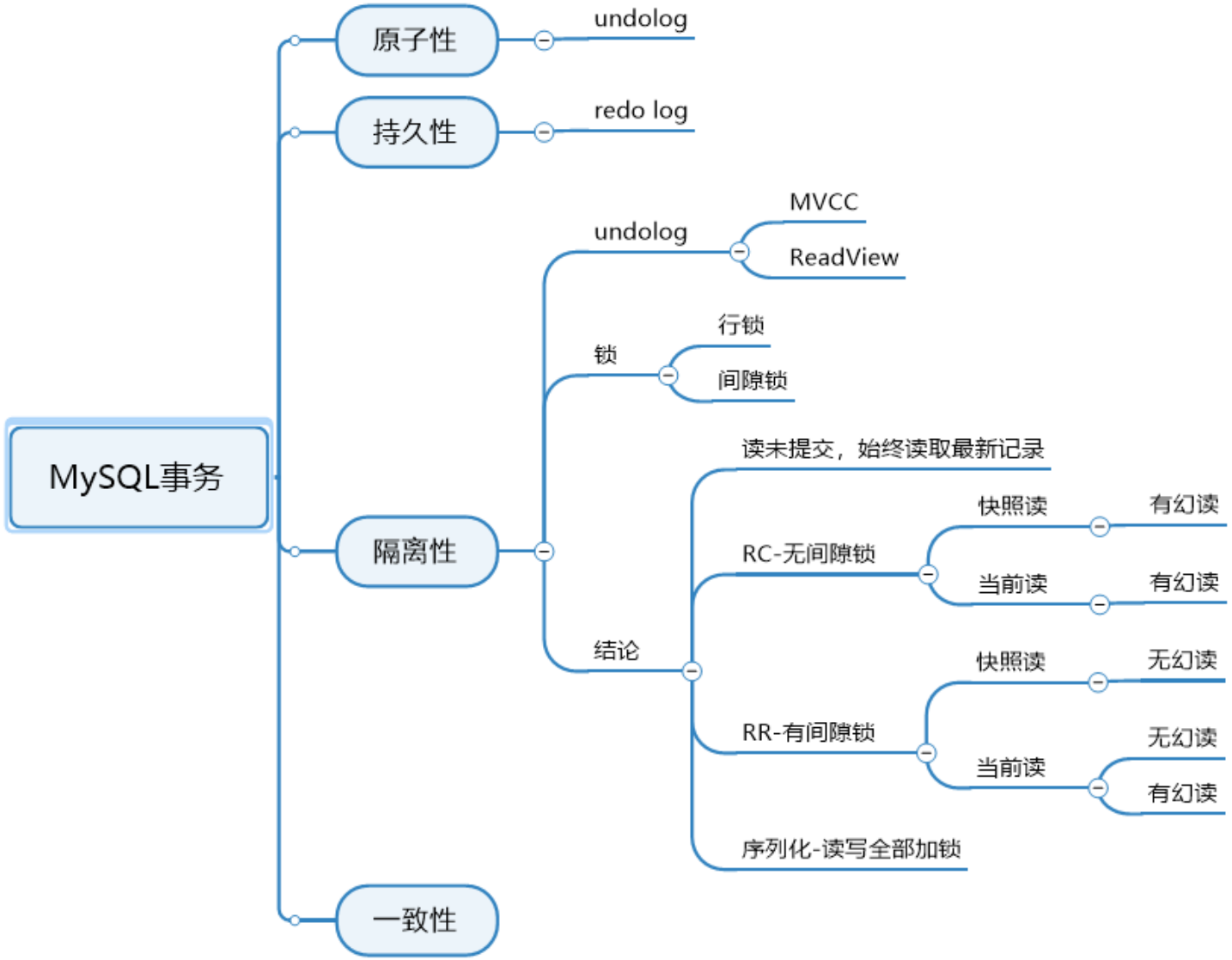
MySQL/InnoDB 事务隔离级别分享

孙志强
2021/05

主要内容


MySQL事务
原子性
undolog
持久性
redo log
隔离性
undolog
MVCC
ReadView
锁
行锁
间隙锁
结论
读未提交，始终读取最新记录
RC-无间隙锁
快照读
有幻读
当前读
有幻读
RR-有间隙锁
快照读
无幻读
当前读
无幻读
有幻读
序列化-读写全部加锁
一致性

主要内容

事务特性

- 原子性（Atomicity
- 一致性（Consistency）
- 隔离性（Isolation）
- 持久性（Durability

日志体系-更新语句的执行过程

- redo log
- binlog

事务隔离

- 隔离性遇见的问题
- 隔离级别相关技术

  四种隔离级别

  一致性视图(read view)

  当前读

  快照读

  隔离性实现

  undo log

  可见性分析

  锁

总结

目标

- 事物隔离级别，带有多事物演示案例，来说明种隔离级别下对脏读 可重复 患读 的不同制约
- redolog binlog undolog几种日志的作用
- mvcc实现原理 目的 以及多事物并行时案例

准备工作

MySQL版本- 5.7.17
准备素材:表:

```
CREATE TABLE `tbl` (  `id` int(11) NOT NULL AUTO_INCREMENT,  `name` varchar(255) DEFAULT NULL,  `status` int(10) DEFAULT NULL,  `is_delete` int(4) DEFAULT NULL,  PRIMARY KEY (`id`),  KEY `idx_status` (`status`)) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8;
```

数据:

```
INSERT INTO `tbl`(`id`, `name`, `status`, `is_delete`) VALUES (1, '张三', 1, 0);
INSERT INTO `tbl`(`id`, `name`, `status`, `is_delete`) VALUES (3, '1', 1, 0);
```

主要内容

事务特性

- 原子性（Atomicity
- 一致性（Consistency）
- 隔离性（Isolation）
- 持久性（Durability

日志体系-更新语句的执行过程

- redo log
- binlog

事务隔离

- 隔离性遇见的问题
- 隔离级别相关技术

  四种隔离级别

  一致性视图(read view)

  当前读

  快照读

  隔离性实现

  undo log

  可见性分析

  锁

总结

事务特性

事务：最小的不可再分的工作单元；通常一个事务对应一个完整的业务(例如银行账户转账业务，该业务就是一个最小的工作单元)

- 原子性(A)：事务是最小单位，不可再分（更多关注多行）
- 一致性(C)：事务要求所有的DML语句操作的时候，必须保证同时成功或者同时失败
- 隔离性(I)：事务A和事务B之间具有隔离性
- 持久性(D)：是事务的保证，事务终结的标志(内存的数据持久到硬盘文件中)

其中：持久性依赖redo log
隔离性+原子性 依赖undolog
最终保证了一致性

主要内容

事务特性

- 原子性（Atomicity
- 一致性（Consistency）
- 隔离性（Isolation）
- 持久性（Durability

日志体系-更新语句的执行过程

- redo log
- binlog

事务隔离

- 隔离性遇见的问题
- 隔离级别相关技术

  四种隔离级别

  一致性视图(read view)

  当前读

  快照读

  隔离性实现

  undo log

  可见性分析

  锁

总结

redo log

作用：确保事务的持久性，故障恢复。

- 先写日志，再写磁盘（write-ahead logging ）

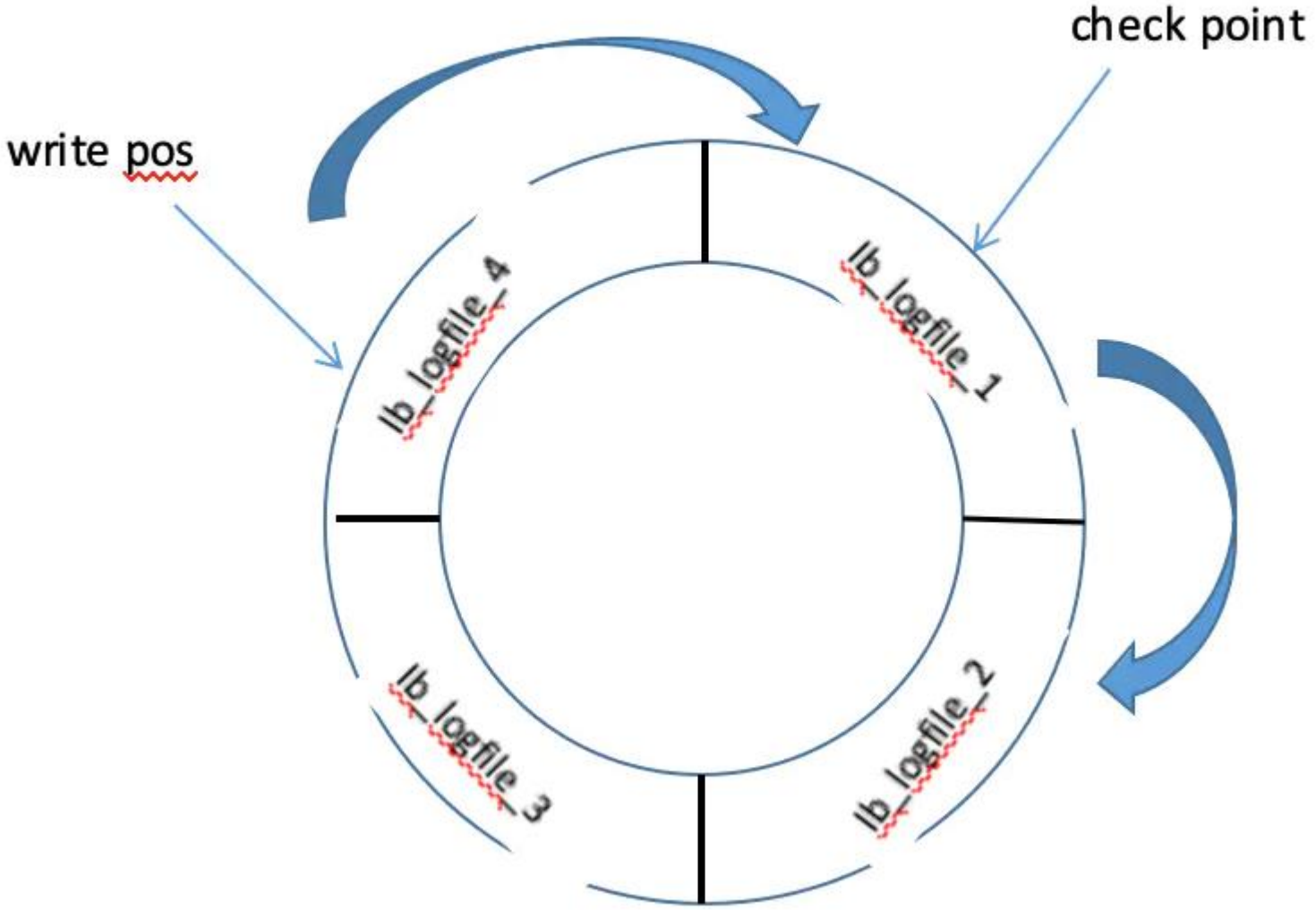
当有一条记录需要更新的时候，InnoDB引擎就会先把记录写到redo log里面，并更新内存，这个时候更新就算完成了。同时，InnoDB引擎会在适当的时候，将这个操作记录更新到磁盘里面

- redo log记录方式

redolog的大小是固定的innodb_log_files_in_grou和innodb_log_file_size配置日志文件数量和每个日志文件大小，redo log采用循环写的方式记录，当写到结尾时，会回到开头循环写日志 。

比如可以配置

innodb_log_files_in_group=4
innodb_log_file_size=1GB
那么整个redo log的结构，就相当于有四个文件组成，每个文件大小事1GB，从头开始写，写到末尾循环写，如右图



https://www.jianshu.com/p/4bcfffb27ed5

binlog 日志

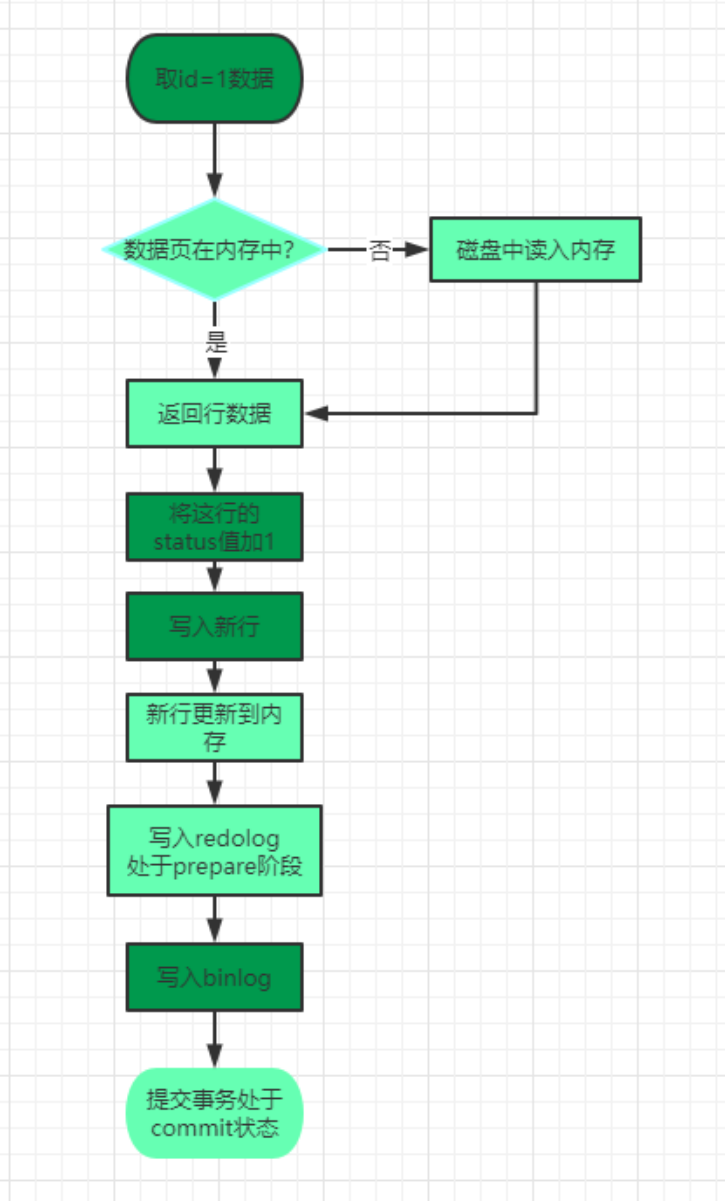
redo log是InnoDB引擎特有的日志，而Server层也有自己的日志，称为binlog日志（归档日志）

- binlog是server层实现的，意味着所有引擎都可以使用binlog日志
- binlog通过追加的方式写入的，可通过配置参数max_binlog_size设置每个binlog文件的大小，当文件大小大于给定值后，日志会发生滚动，之后的日志记录到新的文件上。
- binlog有3种记录模式，statement格式的话是记sql语句， row格式会记录行的内容，记两条，更新前和更新后都有，第3种混合使用。
- binlog常用于主从复制，数据同步等场景，京东有binlog组件，叫做binlog平台。大体场景是用于无业务侵入，数据变更发送mq同步其他业务的目的，保证了数据一致性。

redo log对比binlog

- redo log是InnoDB引擎特有的；binlog是MySQL的Server层实现的，所有引擎都可以使用。
- redo log是物理日志，记录的是“在某个数据页上做了什么修改”；binlog是逻辑日志，记录的是这个语句的原始逻辑，比如“给ID=2这一行的c字段加1 ”。
- redo log是循环写的，空间固定会用完；binlog是可以追加写入的。“追加写”是指binlog文件写到一定大小后会切换到下一个，并不会覆盖以前的日志。

一条更新语句的执行过程



redo log写入拆成了两个步骤：prepare和commit，这就是大名鼎鼎的两阶段提交。

深色部分代表Innodb内部流程，浅色部分代表service层流程

两阶段提交就是为了保证binlog内容和redo log内容一致。

redo log：保证了数据持久性

主要内容

事务特性

- 原子性（Atomicity
- 一致性（Consistency）
- 隔离性（Isolation）
- 持久性（Durability

日志体系-更新语句的执行过程

- redo log
- binlog

事务隔离

- 隔离性遇见的问题
- 隔离级别相关技术

  四种隔离级别

  一致性视图(read view)

  当前读

  快照读

  隔离性实现

  undo log

  可见性分析

  锁

总结

隔离性遇见的问题

•脏读:此情况仅会发生在： 读未提交的的隔离级别.

当数据库中一个事务A正在修改一个数据但是还未提交或者回滚，另一个事务B 来读取了修改后的内容并且使用了，之后事务A提交了，此时就引起了脏读。

•不可重复读：此情况仅会发生在：读未提交、读提交的隔离级别.

在一个事务A中多次操作数据，在事务操作过程中(未最终提交)，事务B也才做了处理，并且该值发生了改变，这时候就会导致A在事务操作的时候，发现数据与第一次不一样了。 就是不可重复读。

•幻读：此情况会回发生在：读未提交、读提交、可重复读的隔离级别

一个事务按相同的查询条件重新读取以前检索过的数据，却发现其他事务插入了满足其查询条件的新数据，这种现象就称为幻读。

幻读是指当事务不是独立执行时发生的一种现象，例如第一个事务对一个表中的数据进行了修改，比如这种修改涉及到表中的“全部数据行”。同时，第二个事务也修改这个表中的数据，这种修改是向表中插入“一行新数据”。那么，以后就会发生操作第一个事务的用户发现表中还存在没有修改的数据行，就好象发生了幻觉一样.

隔离级别验证

READ UNCOMMITTED： 出现脏读

session1	session2	session3
set autocommit = 0;	set autocommit = 0;	
set session TRANSACTION ISOLATION level READ UNCOMMITTED ;	set session TRANSACTION ISOLATION level READ UNCOMMITTED ;	
BEGIN ;	BEGIN ;	
	select * from tbl where id =1 ;	select * from tbl where id =1 ;
update tbl set name =‘wangwu' where id = 1 ;		
	select * from tbl where id =1 ;	select * from tbl where id =1 ;
COMMIT ;	COMMIT ;	

READ COMMITTED：解决了脏读，存在不可重复读

session1	session2	session3
set autocommit = 0;	set autocommit = 0;	
set session TRANSACTION ISOLATION level READ COMMITTED ;	set session TRANSACTION ISOLATION level READ COMMITTED ;	
BEGIN ;	BEGIN ;	
	select * from tbl where id =1 ;	select * from tbl where id =1 ;
update tbl set name =‘wangwu' where id = 1 ;		
	select * from tbl where id =1 ;	select * from tbl where id =1 ;
COMMIT ;		
	COMMIT ;	

REPEATABLE READ：解决了不可重复读

session1	session2	session3
set autocommit = 0;	set autocommit = 0;	
set session TRANSACTION ISOLATION level REPEATABLE READ ;	set session TRANSACTION ISOLATION level REPEATABLE READ ;	
BEGIN ;	BEGIN ;	
	select * from tbl where id =1 ;	select * from tbl where id =1 ;
update tbl set name =‘wangwu' where id = 1 ;		
COMMIT ;		
	select * from tbl where id =1 ;	select * from tbl where id =1 ;
	COMMIT ;	

REPEATABLE READ：存在幻读

session1	session2
set autocommit = 0;	set autocommit = 0;
set session TRANSACTION ISOLATION level REPEATABLE READ ;	set session TRANSACTION ISOLATION level REPEATABLE READ ;
BEGIN ;	BEGIN ;
	select * from tbl where id =2 ;
insert into tbl(id,name) VALUES(2,'wangwu') ;	
COMMIT ;	
	update tbl set name ='wangwu' ;
	select * from tbl where id =2 ;
	COMMIT ;

主要内容

事务特性

- 原子性（Atomicity
- 一致性（Consistency）
- 隔离性（Isolation）
- 持久性（Durability

日志体系-更新语句的执行过程

- redo log
- binlog

事务隔离

- 隔离性遇见的问题
- 隔离级别相关技术

  四种隔离级别

  一致性视图(read view)

  当前读

  快照读

  隔离性实现

  undo log

  可见性分析

  锁

总结

隔离级别

•读未提交（READ UNCOMMITTED）
可以读取未提交记录。此隔离级别，不会使用，忽略

•读提交（READ COMMITTED）：
1.快照读避免脏读，会有不可重复读
2.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。

•可重复读(REPEATABLE READ)：
1. 快照读避免不可重复读，会有幻读。
2.针对当前读，RR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，新事物的满足查询条件的记录不能够插入（间隙锁，需要等待前一事务提交才能插入），在一定程度上避免了幻读现象。但是无法完全避免幻读，比如前一事务没有锁柱另一事务的insert，而且前一事务变成当前读了新数据。

•串行化(SERIALIAZABLE)：
从MVCC并发控制退化为基于锁的并发控制。不区分快照读与当前读。所有的读操作均为当前读，读都加锁（S锁），写都加锁（X锁）

SERIALIAZABLE 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

隔离性增高，性能降低，安全性增高， MySQL数据库默认 REPEATABLE READ

主要内容

事务特性

- 原子性（Atomicity
- 一致性（Consistency）
- 隔离性（Isolation）
- 持久性（Durability

日志体系-更新语句的执行过程

- redo log
- binlog

事务隔离

- 隔离性遇见的问题
- 隔离级别相关技术

  四种隔离级别

  一致性视图(read view)

  当前读

  快照读

  隔离性实现

  undo log

  可见性分析

  锁

总结

几个概念

一致性读视图	即consistent read view，用于支持RC（Read Committed，读提交）和RR（Repeatable Read，可重复读）隔离级别的实现。
当前读	读取的是记录的最新版本，读取时还要保证其他并发事务不能修改当前记录，会对读取的记录进行加锁。 示例：select … lock in share mode; select … for update ; update, insert ,delete
快照读	不加锁的select操作就是快照读，即不加锁的非阻塞读； 示例：简单的select操作(不包括 select … lock in share mode, select … for update)

事实上，数据库里面会创建一个视图，访问的时候以视图的逻辑结果为准，

- “可重复读”隔离级别下，这个视图是在事务启动时创建的，整个事务存在期间都用这个视图。
- “读提交”隔离级别下，这个视图是在每个SQL语句开始执行的时候 创建的。
- “读未提交”隔离级别下直接返回记录上的最新值，没有视图概念；
- “串行化”隔离级别下直接用加锁的方式来避免并行访问。

主要内容

事务特性

- 原子性（Atomicity
- 一致性（Consistency）
- 隔离性（Isolation）
- 持久性（Durability

日志体系-更新语句的执行过程

- redo log
- binlog

事务隔离

- 隔离性遇见的问题
- 隔离级别相关技术

  四种隔离级别

  一致性视图(read view)

  当前读

  快照读

  隔离性实现

  undo log

  可见性分析

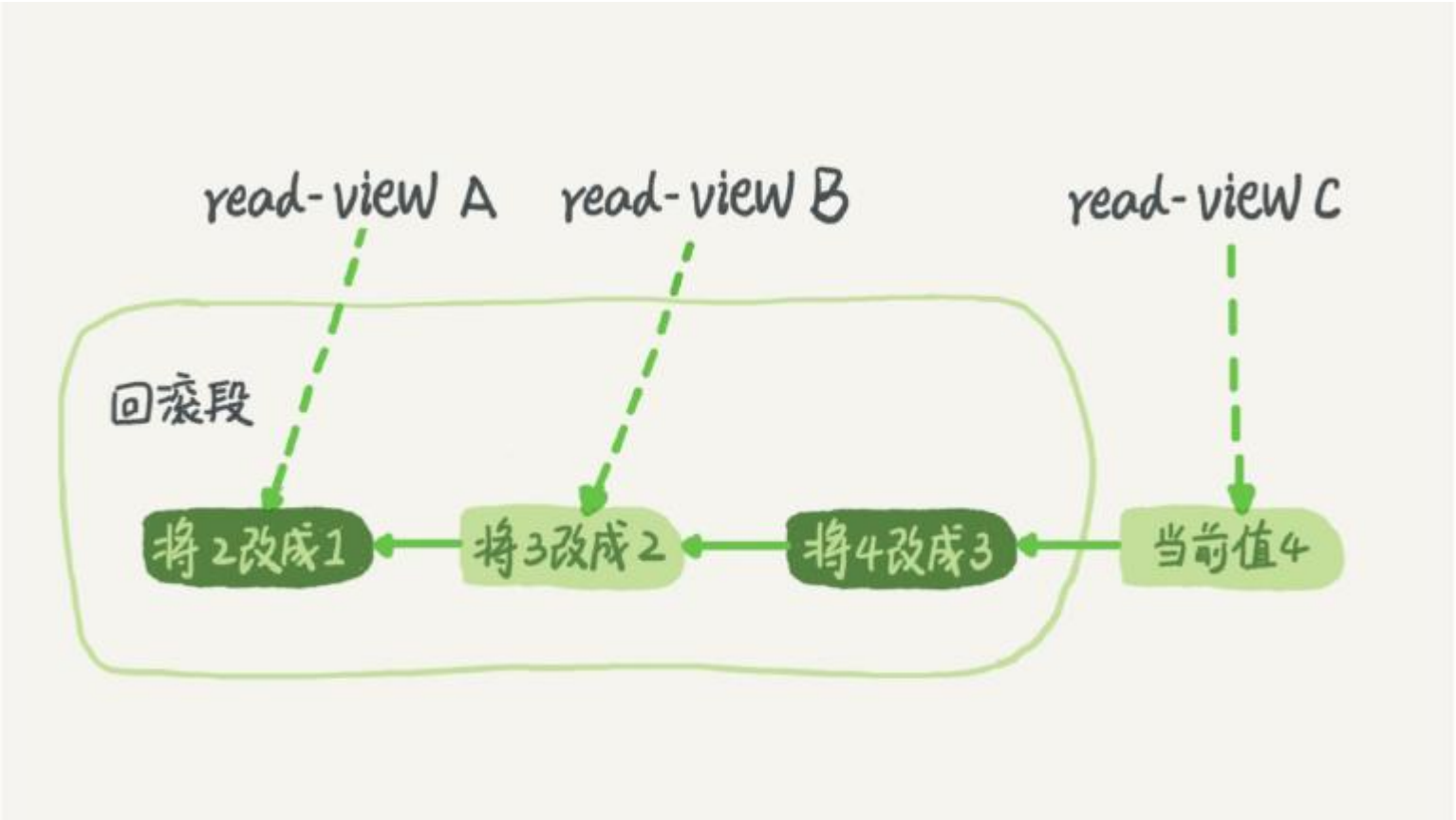
  锁

总结

隔离性实现

在MySQL中，在每一次更新的时候，都会记录一条回滚操作，记录上最新的值，通过回滚操作，都可以得到前一个状态的值。

比如一个值从1，被顺序改成了2、3、4，在回滚日志里面就会有类似下面的记录：



- 当前值是4
- 不同时刻启动的事务会有不同的read-view。
- 在视图A、B、C里面，这一个记录的值分别是1、2、4，同一条记录在系统中可以存在多个版本，就是数据库的多版本并发控制（MVCC）。
- 即使现在有另外一个事务正在将4改成5，这个事务跟read-view A、B、C对应的事务是不会冲突的。

https://blog.csdn.net/u010002184/article/details/88526708

隔离性实现

RR级别下：我们继续看一组案例：

开始时：status=1

session1	session2	session3
start transaction with consistent snapshot;	start transaction with consistent snapshot;	
		update tbl set status =status+1 where id = 3 ;
	update tbl set status =status+1 where id = 3 ;	
select status from tbl where id =3 ;	select status from tbl where id =3 ;	
COMMIT ;	COMMIT ;	

问题：

1.session2里面查询的结果，status=3
2.session1里面查询的结果，status=1

结论与你了解的事务隔离是否矛盾？

主要内容

事务特性

- 原子性（Atomicity
- 一致性（Consistency）
- 隔离性（Isolation）
- 持久性（Durability

日志体系-更新语句的执行过程

- redo log
- binlog

事务隔离

- 隔离性遇见的问题
- 隔离级别相关技术

  四种隔离级别

  一致性视图(read view)

  当前读

  快照读

  隔离性实现

  undo log

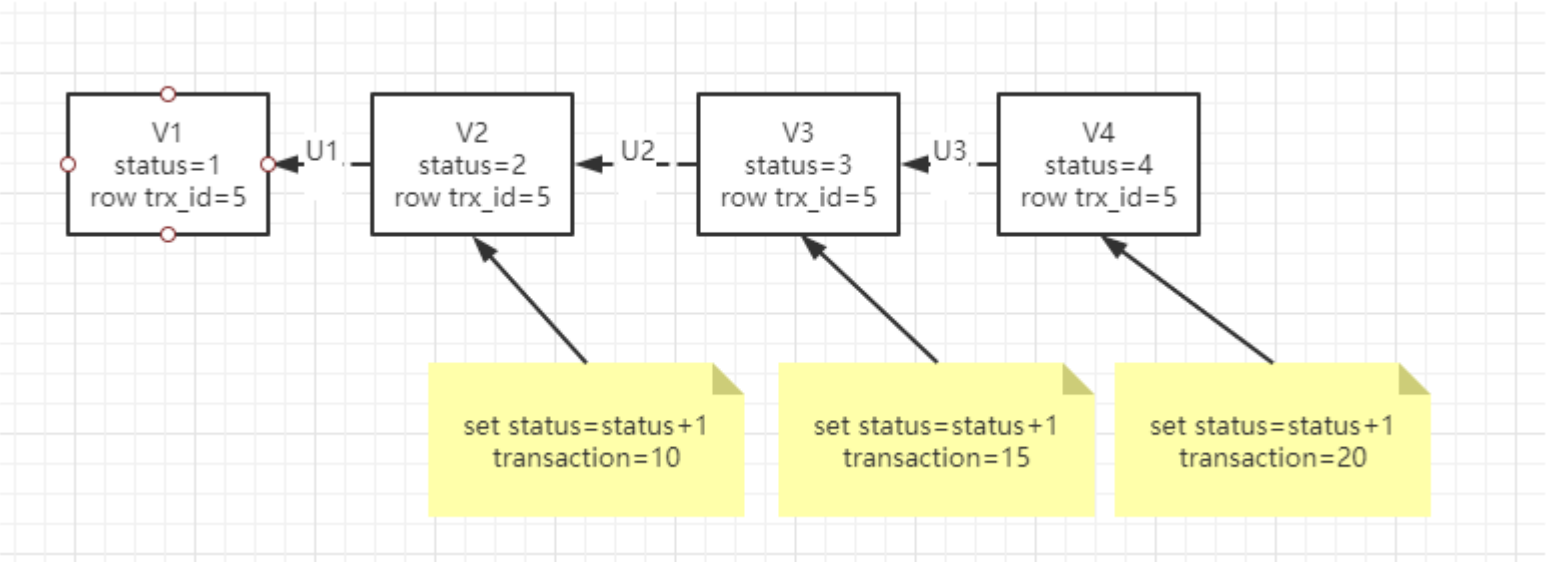
  可见性分析

  锁

总结

undo log

前文我们说过 回滚日志，何为回滚日志(undo log)，在哪里?

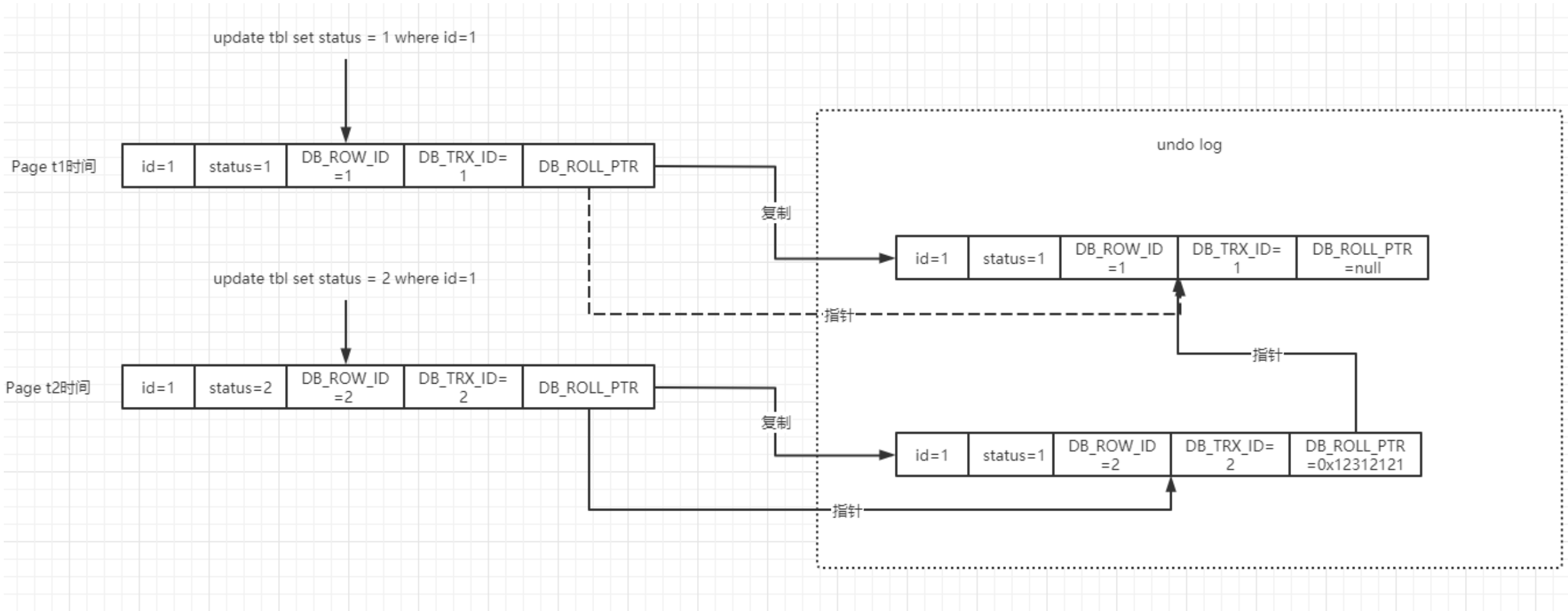


实际上：上图中三个虚线箭头，就是undo log，而且，V1，V2，V3并不是物理上真实存在的，而是每次需要的时候根据当前的版本和undo log计算出来的。

比如需要V2，就是通过V4依次执行U3，U2计算出来。

其中V，我们成为Read View（读视图）

undo log结构原理



修改数据时，先复制数据到undo log
所有历史的事务修改，会在Undo log中形成链表，undo log的链首就是最新的旧记录

主要内容

事务特性

- 原子性（Atomicity
- 一致性（Consistency）
- 隔离性（Isolation）
- 持久性（Durability

日志体系-更新语句的执行过程

- redo log
- binlog

事务隔离

- 隔离性遇见的问题
- 隔离级别相关技术

  四种隔离级别

  一致性视图(read view)

  当前读

  快照读

  隔离性实现

  undo log

  可见性分析

  锁

总结

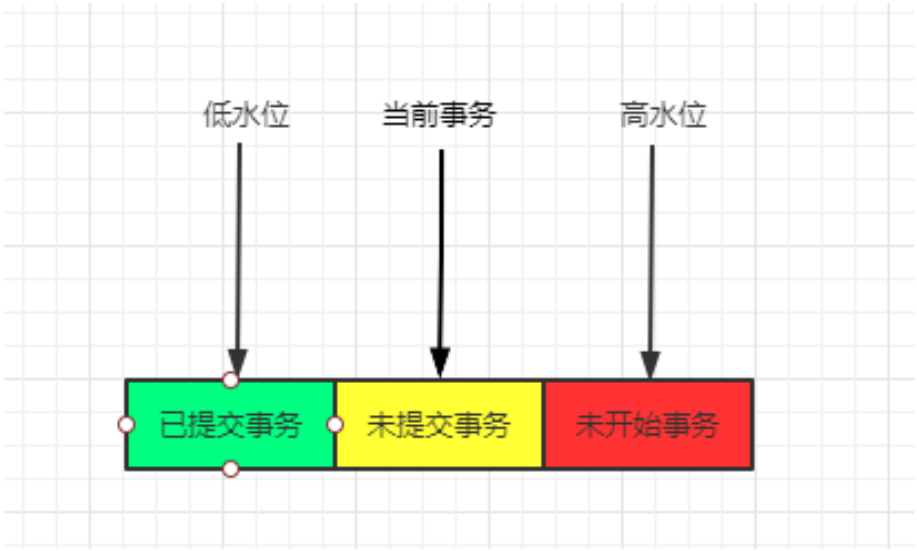
可见性分析

上图里面，其中DB_TRX_ID 代表当前事务ID（每个事务开启，都会被分配，递增）

前文Read View，是另一个维度，维护了一组DB_TRX_ID，组成：

1. 已提交事务（trx_id 最小值）；
2. 当前事务（未提交事务集合）；
3. 未开始事务（目前最大trx_id+1,下次分配用）

可见性规则如下：将要被修改的数据的最新记录中的DB_TRX_ID（即当前事务ID）取出来。昨日下规则对比：



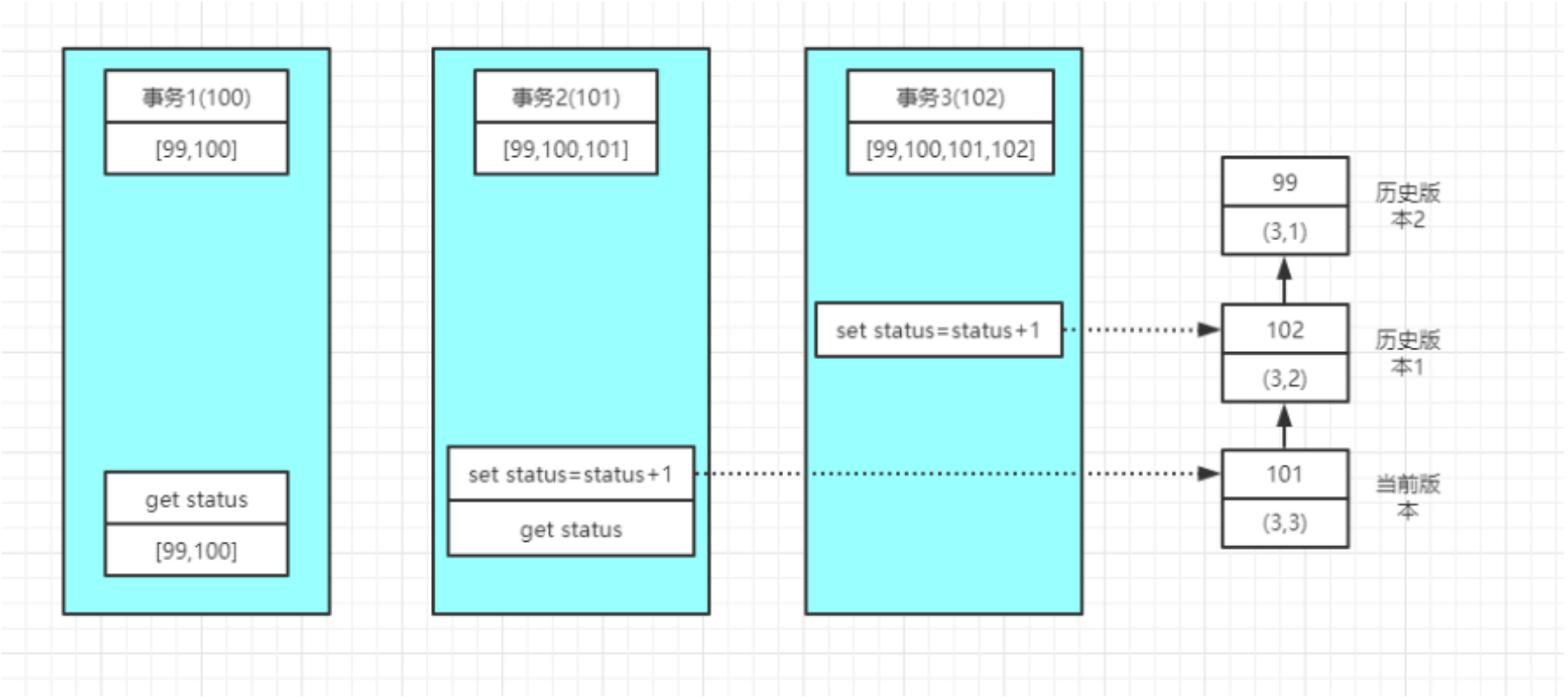
1. 如果落在绿色部分，表示这个版本是已提交的事务或者是当前事务自己生成的，这个数据是可见的；
2. 如果落在红色部分，表示这个版本是由将来启动的事务生成的，是肯定不可见的；
3. 如果落在黄色部分，那就包括两种情况
   a. 若 row trx_id在数组中，表示这个版本是由还没提交的事务生成的，不可见；
   b. 若 row trx_id不在数组中，表示这个版本是已经提交了的事务生成的，可见。

结论：

- 一个事务只需要在启动的时候声明说，“以我启动的时刻为准，如果一个数据版本是在我启动之前生成的，就认；如果是我启动以后才生成的，我就不认，我必须要找到它的上一个版本”。
- 如果“上一个版本”也不可见，那就得继续往前找。还有，如果是这个事务自己更新的数据，它自己还是要认的

可见性分析-session1分析

我们先分析session1中的status为什么等于1



1. 事务1开始前，假设系统里面只有一个活跃事务ID是99；

2. 事务1、2、3的版本号分别是100、101、102，且当前系统里只有这四个事务；

3. 三个事务开始前，(3,1）这一行数据的row trx_id是99。

这样，
事务1的视图数组： [99,100],
事务2的视图数组：[99,100,101],
事务3的视图数组：[99,100,101,102]。

如果现在事务1要来读数据了，它的视图数组是[99,100]。当然了，读数据都是从当前版本读起的。所以，事务1查询语句的读数据流程是这样的：

- 找到(3,3)的时候，判断出row trx_id=101，比高水位大，处于红色区域，不可见；
- 接着，找到上一个历史版本，一看row trx_id=102，比高水位大，处于红色区域，不可见；
- 再往前找，终于找到了（1,1)，它的row trx_id=99，比低水位小，处于绿色区域，可见。

这样执行下来，虽然期间这一行数据被修改过，但是事务A不论在什么时候查询，看到这行数据的结果都是一致的，所以我们称之为一致性读。

可见性分析- session2分析

如果前面所说的，都是正确的，按照一致性读，事务2的update语句，结果是否有问题?

如果事务2的视图数组是生成的，之后事务3才提交，不是应该看不见(3,2)么，那么接下来的查询怎么会是(3,3)呢

这里就用到了这样一条规则：更新数据都是先读后写的，而这个读，只能读当前的值，称为“当前读”（current read）。

因此，在更新的时候，当前读拿到的数据是(3,2)，更新后生成了新版本的数据3,3)，这个新版本的row trx_id是101。

这里我们提到了一个概念，叫作当前读。其实，除了update语句外，select语句如果加锁，也是当前读。

此时我们修改一下session3，前边的案例是session立马提交，如果不是这样的，结果会怎么样呢？

session1	session2	session3
start transaction with consistent snapshot;	start transaction with consistent snapshot;	start transaction with consistent snapshot;
		update tbl set status =status+1 where id = 3 ;
	update tbl set status =status+1 where id = 3 ;	
select status from tbl where id =3 ;	select status from tbl where id =3 ;	
		commit;
COMMIT ;	COMMIT ;	

这里涉及另外一块知识 锁的概念：“两阶段锁协议”。

两阶段锁协议：行锁是在需要的时候才加上的，但并不是不需要了就立刻释放，而是要等到事务结束时才释放。锁的概念不在本次分享范畴，我们暂时知道有个这样的规则即可。
事务3没提交，也就是说(3,2)这个版本上的写锁还没释放。而事务2是当前读，必须要读最新版本，而且必须加锁，因此就被锁住了，必须等到事务2释放这个锁，才能继续它的当前读。

主要内容

事务特性

- 原子性（Atomicity
- 一致性（Consistency）
- 隔离性（Isolation）
- 持久性（Durability

日志体系-更新语句的执行过程

- redo log
- binlog

事务隔离

- 隔离性遇见的问题
- 隔离级别相关技术

  四种隔离级别

  一致性视图(read view)

  当前读

  快照读

  隔离性实现

  undo log

  可见性分析

  锁

总结

加锁汇总

MySQL加锁，是通过在索引上加锁实现的
如果有索引 以下操作

select * from table where ? Lock in share mode;
select * from table where ? for update;
Insert
update
Delete

1.主键索引
RC、RR给主键索引加行锁

2.唯一索引

RC、RR给唯一索引加行锁，给对应的主键索引加行锁

3.普通索引
如果 RC，普通索引加行锁，对应的逐渐索引加行锁
如果RR，普通索引加行锁，以及这些行锁数据的间隙加范围锁。主键索引对应行加锁

主要内容

事务特性

- 原子性（Atomicity
- 一致性（Consistency）
- 隔离性（Isolation）
- 持久性（Durability

日志体系-更新语句的执行过程

- redo log
- binlog

事务隔离

- 隔离性遇见的问题
- 隔离级别相关技术

  四种隔离级别

  一致性视图(read view)

  当前读

  快照读

  隔离性实现

  undo log

  可见性分析

  锁

总结

总结

至此为止：我们详细的分析了一条更新语句的各种场景，总结一下事物的可重复读的实现原理：

1. 可重复读的核心就是一致性读（consistent read）；而事务更新数据的时候，只能用当前读。如果当前的记录的行锁被其他事务占用的话，就需要进入锁等待。

2. 而读提交的逻辑和可重复读的逻辑类似，它们最主要的区别是：

   - 在可重复读隔离级别下，只需要在事务开始的时候创建一致性视图，之后事务里的其他查询都共用这个一致性视图；
   - 在读提交隔离级别下，每一个语句执行前都会重新算出一个新的视图。

session1和session2的过程加上并发写的分析过程，即为MVCC（数据库的多版本并发控制）原理。

快照-RC-有幻读

session1	session2
set autocommit = 0;	set autocommit = 0;
set session TRANSACTION ISOLATION level READ COMMITTED ;	set session TRANSACTION ISOLATION level READ COMMITTED ;
	BEGIN ;
	select * from tbl where status =2 ;
BEGIN ;	
insert into tbl(id,status) VALUES(5,2) ;	
COMMIT ;	select * from tbl where status =2 ;
	COMMIT ;

快照- RR-无幻读

session1	session2
set autocommit = 0;	set autocommit = 0;
set session TRANSACTION ISOLATION level REPEATABLE READ ;	set session TRANSACTION ISOLATION level REPEATABLE READ ;
	BEGIN ;
	select * from tbl where status =2 ;
BEGIN ;	
insert into tbl(id,status) VALUES(5,2) ;	
COMMIT ;	select * from tbl where status =2 ;
	COMMIT ;

当前读-RC-有幻读

session1	session1
set autocommit = 0;	set autocommit = 0;
set session TRANSACTION ISOLATION level READ COMMITTED ;	set session TRANSACTION ISOLATION level READ COMMITTED ;
	BEGIN ;
	select * from tbl where status =2 ; update name='' where status =2 ;
BEGIN ;	
insert into tbl(id,status) VALUES(5,2) ; COMMIT ;	
	select * from tbl where status =2 ;
	COMMIT ;

当前读-RR-无幻读

session1	session1
set autocommit = 0;	set autocommit = 0;
set session TRANSACTION ISOLATION level REPEATABLE READ ;	set session TRANSACTION ISOLATION level REPEATABLE READ ;
	BEGIN ;
	select * from tbl where status =2 ; update tbl set name='' where status =2 ;
BEGIN ;	
insert into tbl(id,status) VALUES(5,2) ; COMMIT ;	
	select * from tbl where status =2 ;
	COMMIT ;

当前读-RR-有幻读

session1	session2
set autocommit = 0;	set autocommit = 0;
set session TRANSACTION ISOLATION level REPEATABLE READ ;	set session TRANSACTION ISOLATION level REPEATABLE READ ;
	BEGIN ;
BEGIN ;	select * from tbl where status =2 ;
insert into tbl(id,status) VALUES(5,2) ; COMMIT ;	
	update name='' where status =2 ;
	select * from tbl where status =2 ;
	COMMIT ;

THANKS