知乎

一文理解Mysql MVCC

强哥叨逼叨
愿你走过一生，归来仍是少年

关注他

280 人赞同了该文章

MVCC

就是多版本并发控制。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。

为什么需要MVCC呢？数据库通常使用锁来实现隔离性。最原生的锁，锁住一个资源后会禁止其他任何线程访问同一个资源。但是很多应用的一个特点都是读多写少的场景，很多数据的读取次数远大于修改的次数，而读取数据间互相排斥显得不是很必要。所以就使用了一种读写锁的方法，读锁和读锁之间不互斥，而写锁和写锁、读锁都互斥。这样就很大提升了系统的并发能力。之后人们发现并发读还是不够，又提出了能不能让读写之间也不冲突的方法，就是读取数据时通过一种类似快照的方式将数据保存下来，这样读锁就和写锁不冲突了，不同的事务session会看到自己特定版本的数据。当然快照是一种概念模型，不同的数据库可能用不同的方式来实现这种功能。

InnoDB与MVCC

MVCC只在 READ COMMITTED 和 REPEATABLE READ 两个隔离级别下工作。其他两个隔离级别够和MVCC不兼容, 因为 READ UNCOMMITTED 总是读取最新的数据行, 而不是符合当前事务版本的数据行。而 SERIALIZABLE 则会对所有读取的行都加锁。

Redo log, bin log, Undo log

InnoDB中通过undo log实现了数据的多版本，而并发控制通过锁来实现。

知乎

binlog，是mysql服务层产生的日志，常用来进行数据恢复、数据库复制，常见的mysql主从架构，就是采用slave同步master的binlog实现的, 另外通过解析binlog能够实现mysql到其他数据源（如ElasticSearch)的数据复制。

redo log记录了数据操作在物理层面的修改，mysql中使用了大量缓存，缓存存在于内存中，修改操作时会直接修改内存，而不是立刻修改磁盘，当内存和磁盘的数据不一致时，称内存中的数据为脏页(dirty page)。为了保证数据的安全性，事务进行中时会不断的产生redo log，在事务提交时进行一次flush操作，保存到磁盘中, redo log是按照顺序写入的，磁盘的顺序读写的速度远大于随机读写。当数据库或主机失效重启时，会根据redo log进行数据的恢复，如果redo log中有事务提交，则进行事务提交修改数据。这样实现了事务的原子性、一致性和持久性。

undo log: 除了记录redo log外，当进行数据修改时还会记录undo log，undo log用于数据的撤回操作，它记录了修改的反向操作，比如，插入对应删除，修改对应修改为原来的数据，通过undo log可以实现事务回滚，并且可以根据undo log回溯到某个特定的版本的数据，实现MVCC。

版本链与Undo log

innodb中通过B+树作为索引的数据结构，并且主键所在的索引为ClusterIndex(聚簇索引), ClusterIndex中的叶子节点中保存了对应的数据内容。一个表只能有一个主键，所以只能有一个聚簇索引，如果表没有定义主键，则选择第一个非NULL唯一索引作为聚簇索引，如果还没有则生成一个隐藏id列作为聚簇索引。

除了Cluster Index外的索引是Secondary Index(辅助索引)。辅助索引中的叶子节点保存的是聚簇索引的叶子节点的值。

InnoDB行记录中除了刚才提到的rowid外，还有trx_id和db_roll_ptr, trx_id表示最近修改的事务的id,db_roll_ptr指向undo segment中的undo log。

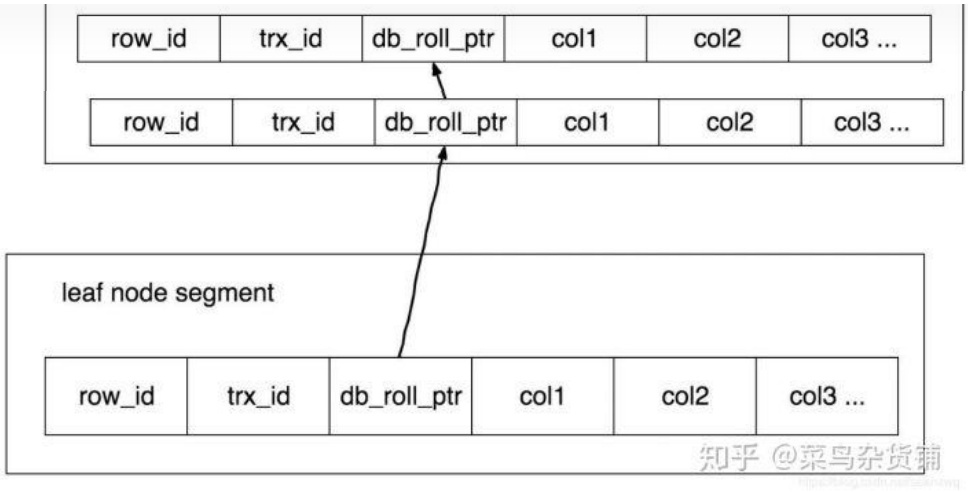
新增一个事务时事务id会增加，trx_id能够表示事务开始的先后顺序。

Undo log分为Insert和Update两种，delete可以看做是一种特殊的update，即在记录上修改删除标记。

update undo log记录了数据之前的数据信息，通过这些信息可以还原到之前版本的状态。

当进行插入操作时，生成的Insert undo log在事务提交后即可删除，因为其他事务不需要这个undo log。

进行删除修改操作时，会生成对应的undo log，并将当前数据记录中的db_roll_ptr指向新的undo log。



ReadView

说完了undo log我们再来看看ReadView。已提交读和可重复读的区别就在于它们生成ReadView的策略不同。

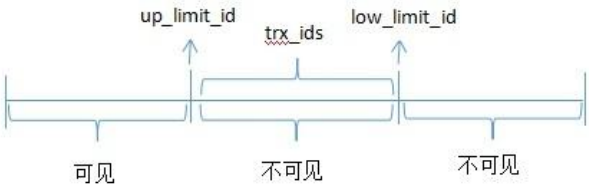
ReadView中主要就是有个列表来存储我们系统中当前活跃着的读写事务，也就是begin了还未提交的事务。通过这个列表来判断记录的某个版本是否对当前事务可见。其中最主要的与可见性相关的属性如下：

up_limit_id：当前已经提交的事务号 + 1，事务号 < up_limit_id ，对于当前Read View都是可见的。理解起来就是创建Read View视图的时候，之前已经提交的事务对于该事务肯定是可见的。

low_limit_id：当前最大的事务号 + 1，事务号 >= low_limit_id，对于当前Read View都是不可见的。理解起来就是在创建Read View视图之后创建的事务对于该事务肯定是不可见的。

trx_ids：为活跃事务id列表，即Read View初始化时当前未提交的事务列表。所以当进行RR读的时候，trx_ids中的事务对于本事务是不可见的（除了自身事务，自身事务对于表的修改对于自己当然是可见的）。理解起来就是创建RV时，将当前活跃事务ID记录下来，后续即使他们提交对于本事务也是不可见的。

用一张图更好的理解一下：



Read View 可见性展示

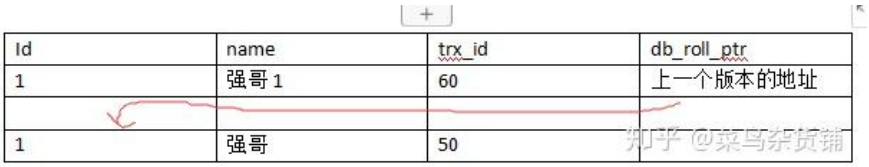
最后我们来举个例子让我们更好理解上面的内容。

比如我们有如下表：

Id	name	trx_id	db_roll_ptr
1	强哥	50	上个版本的地址

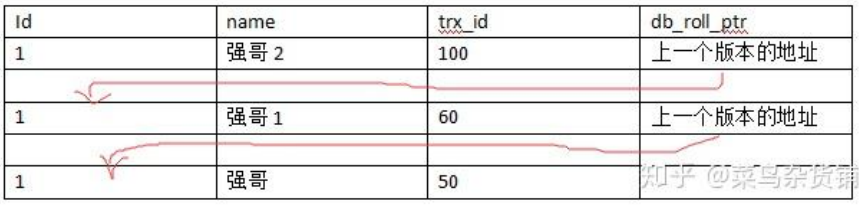
知乎

此时undo log存在版本链如下：



Id	name	trx_id	db_roll_ptr
1	强哥 1	60	上一个版本的地址
1	强哥	50	

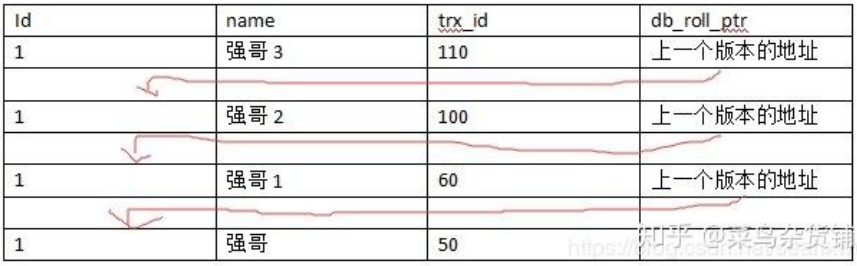
提交事务id是60的记录后，接着有一个事务id为100的事务，修改name=强哥2，但是事务还没提交。则此时的版本链是：



Id	name	trx_id	db_roll_ptr
1	强哥 2	100	上一个版本的地址
1	强哥 1	60	上一个版本的地址
1	强哥	50	

此时另一个事务发起select语句查询id=1的记录，因为trx_ids当前只有事务id为100的，所以该条记录不可见，继续查询下一条，发现trx_id=60的事务号小于up_limit_id，则可见，直接返回结果强哥1。

那这时候我们把事务id为100的事务提交了，并且新建了一个事务id为110也修改id为1的记录name=强哥3，并且不提交事务。这时候版本链就是：



Id	name	trx_id	db_roll_ptr
1	强哥 3	110	上一个版本的地址
1	强哥 2	100	上一个版本的地址
1	强哥 1	60	上一个版本的地址
1	强哥	50	

这时候之前那个select事务又执行了一次查询,要查询id为1的记录。

如果你是已提交读隔离级别READ_COMMITED，这时候你会重新一个ReadView，那你的活动事务列表中的值就变了，变成了[110]。按照上的说法，你去版本链通过trx_id对比查找到合适的结果就是强哥2。

如果你是可重复读隔离级别REPEATABLE_READ，这时候你的ReadView还是第一次select时候生成的ReadView,也就是列表的值还是[100]。所以select的结果是强哥1。所以第二次select结果和第一次一样，所以叫可重复读！

也就是说已提交读隔离级别下的事务在每次查询的开始都会生成一个独立的ReadView,而可重复读隔离级别则在第一次读的时候生成一个ReadView，之后的读都复用之前的ReadView。

这就是Mysql的MVCC,通过版本链，实现多版本，可并发读-写，写-读。通过ReadView生成策略的不同实现不同的隔离级别。

参考：

人类身份验证 - SegmentFault

知乎

何登成的技术博客 " InnoDB多版本(MVCC)实现简要分析

关注公众号获取更多内容，有问题也可在公众号提问哦：

强哥叨逼叨

叨逼叨编程、互联网的见解和新鲜事

发布于 2019-05-23 20:58

MySQL MVCC

推荐阅读

java面试一日一题：讲下mysql中的undolog

问题：请讲下mysql中undo log的作用 分析：mysql中有很多日志，例，bin log undo log redo log，要弄清楚这些日志的作用，就要了解这些日志出现的背景及要解决的问题； 回答要点： 主要从以...

北漂程序员 发表于java面...

MySQL事务处理特性的实现原理

摘要：事务这个词来自于英语中的transactional这个词的翻译，这个词的含义更多的是指 “交易”。在数据库系统或者软件系统中我们通常 称 transactional 为事务事务这个词来自于英语中的tran...

华为云开发... 发表于程序员之家

Mys
undo
我们在
核心日
log、
serve
undo
日志，
勤劳的

34 条评论 切换为时间排序

写下你的评论...

简单 2021-06-11

一个表只能有一个主键，所以只能有一个聚簇索引 。这个因果关系不对吧?

3

已赞同 280 34 条评论 分享 喜欢 收藏 申请转载 ... 02-28

知乎

简单 2021-06-11

辅助索引中的叶子节点保存的是聚簇索引的叶子节点的值。这句话也不太对，保存的应该是聚簇索引的键

8

改变 回复 简单 2021-08-11

准确点是主键ID

2

吃葡萄 2019-12-27

up_limit_id：当前已经提交的事务号 + 1？？如果有事物id100,200,500。事事务id为500的提交了，up_limit_id是501?

1

wol 回复 吃葡萄 2020-04-04

up_limit_id不是当前已经提交的事务号 + 1，而是活跃事务列表中最小的事务号，可以参考：blog.csdn.net/Waves___/...

7

Hello Hawaii 回复 wol 2020-07-05

谢谢,一篇好文

1

展开其他 1 条回复

鹰影 2019-07-17

为什么提交事务id为100后还要新建事务110，如果不新建110的事务，读已提交是不是讲不通？如果不新建110的事务按你说的理论读已提交应该是强哥1(60),而事实上是强哥3(110)。

1

强哥叨逼叨 (作者) 回复 鹰影 2019-07-17

如果没有110事务，而隔离级别是读已提交，这时候因为会重新建一个ReadView，所以应该看到的是强哥2哦（110）。

额，你上面的问题，都没有新建110事务了，还哪里来的强哥3（110）

2

陈小强 2019-07-04

明白， 多谢

1

Spongecaptain 2021-04-22

神棍文章。事务 ID 大小顺序只是 begin 顺序，而不是 commit 顺序，因此作者所说的：“up_limit_id：当前已经提交的事务号 + 1。事务号 < up_limit_id ，对于当前Read View都是可见的。 “是什么逻辑。。。。如果是这样，我只要 begin; commit; 一下，事务全都可见了?

1

光风霁月 2021-03-08

mvcc无法解决幻读吧，mvcc只能实现读已提交和可重复读两个隔离级别

1

2021-08-03

知乎

赞

zero 回复 光风霁月 2021-10-19

InnoDB在可重复度就可以是解决幻读了。

赞

展开其他 3 条回复

岳晓春 2020-04-05

能说说插入时候的 查询吗

赞

夹心面包 回复 岳晓春 2020-06-27

插入的时候查询一样的道理,也是根据的版本号。RR情况下因为有gap lock，形成read view,事务的查询不会没查到,RC情况下就是读已提交,能查到新插入的数据行

赞

奔跑的猪 2019-09-25

很明白

赞

一丈白 2021-10-15

大部分是正确的，有些细节修改一下，仍不失为一篇优秀的技术文章。

赞

简单 2021-06-10

【其他两个隔离级别够和MVCC不兼容】是不兼容还是不需要啊?

赞

普度众生的面瘫青年 2021-05-04

可重复读的级别，一般有什么应用场景呢

赞

魔君不真2 2021-03-30

up_limit_id 定义有误！！！

不是当前已经提交的事务号 + 1，而是活跃事务列表中最小的事务号。

赞

三毛 2021-03-13

mvcc也是可以实现冲突可串行化的，只要在每个Object上维护一个读时间戳，记录读过这个对象的最大时间戳，然后需要每次写的时候检查写事务的时间戳和这个时间戳的大小。如果这个时间戳大，那么就abort当前写事务，否则可以正常创建新版本。

赞

粥米小 2021-02-22

Mvcc如何防止幻读?

赞

长路 回复 粥米小 2021-04-30

应该是间隙锁有效解决了幻读吧

1

2021-02-18

已赞同 280 34 条评论 分享 喜欢 收藏 申请转载 ...

知乎

不懂才觉得高深 2021-01-11

非常棒

赞

兄弟连 2020-12-11

简洁清晰，感谢楼主

赞

知乎用户U75043 2020-11-03

THX，MARK

赞

ZZZZZZJJJJJJ 2020-10-20

懂了懂了👍

赞