Mysql数据库事务的隔离级别和锁的实现原理分析 - 黑伯的博客小站 - 博客频道

分类：
数据库（5）

目录(?)[+]

MySQL数据库的事务隔离级别相信很多同学都知道.

大家有没有想过它是如何实现的呢?带着这些问题我翻阅了相关数据库的书籍和资料,把我的理解写下来.

一：事务隔离级别

mysql数据库的隔离界别如下:

1, READ UNCOMMITTED(未提交读)

事务中的修改,即使没有提交,对其它事务也是可见的. 脏读(Dirty Read).

2, READ COMMITTED(提交读)

一个事务开始时,只能"看见"已经提交的事务所做的修改. 这个级别有时候也叫不可重复读(nonrepeatable read).

3, REPEATABLE READ(可重复读)

该级别保证了同一事务中多次读取到的同样记录的结果是一致的. 但理论上,该事务级别还是无法解决另外一个幻读的问题(Phantom Read).

幻读: 当某个事务读取某个范围内的记录时,另外一个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时,会产生幻行.(Phantom Row).

幻读的问题理应由更高的隔离级别来解决,但mysql和其它数据不一样,它同样在可重复读的隔离级别解决了这个问题.

也就是说, mysql的可重复读的隔离级别解决了 "不可重复读" 和 “幻读” 2个问题. 稍后我们可以看见它是如何解决的.

而oracle数据库,可能需要在 “SERIALIZABLE ” 事务隔离级别下才能解决 幻读问题.

mysql默认的隔离级别也是: REPEATABLE READ(可重复读)

4, SERIALIZABLE (可串行化)

强制事务串行执行,避免了上面说到的 脏读,不可重复读,幻读 三个的问题.

二：MVCC(Multi-Version Concurrency Control) 多版本并发控制

MVCC的实现,是通过保存数据在某个时间点的快照来实现的. InnoDB的MVCC是通过在每行记录后面保存2个隐藏的列来实现的,一列保存了行的创建时间,一列保存了行的过期时间(或删除时间).但它们都存储的是系统版本号MVCC最大的作用是：实现了非阻塞的读操作,写操作也只锁定了必要的行.
MYSQL的MVCC 只在 read committed 和 repeatable read 2个隔离级别下工作.

在MVCC的机制下,mysql InnoDB(默认隔离级别)的增删改查变成了如下模式:

SELECT:
1, InnoDB只查找版本早于当前事务版本的数据行(行的系统版本号小于等于事务的系统版本号)
2, 行的删除号要么未定义,要么大于当前事务版本号,这样可以确保事务读取到的行,在事务开始之前未被删除.

INSERT:
InnoDB 为新插入的每一行保存当前系统版本号做为行版本号。

DELETE:
INNODB 为删除的每一行保存当前系统版本号作为行删除标识

UPDATE:
InnoDB 为插入的每一行新记录,保存当前系统版本号作为行版本号,同时保存当前系统版本号到原来的行作为行删除标识.

注意: 上面的读取方式只在InnoDB默认隔离级别下工作,其它的隔离级别会有很大的差异,稍后会看到.三, InnoDB锁.InnoDB的锁大致分为:3.1 行锁支持并发高,带来最大的锁开销. 在存储引擎层实现,服务器感知不到
3.2 表锁服务器会为诸如: ALTER Table 之类的语句使用表锁,忽略存储引擎的锁机制但锁的类型又分为:(1). 共享锁(S Lock) , 允许事务读取一行数据(2). 排他锁(X Lock),允许事务删除或更新一行数据.

3.3 意向锁

InnoDB还实现了一种锁,叫意向锁(Intention Lock).意向锁是将锁定的对象分为多个层次.意向锁的类型分为:(1). 意向共享锁(IS Lock) (2). 意向排他锁(IX Lock)比如: 需要对页上的记录加X锁,那么需要分别对 数据库A,表,页 上加意向锁IX,最后对记录r上加X锁.一旦对数据库A,表,页上加IX锁失败,则阻塞.四: 一致性非锁定读(consistent nonlocking read) 4.1 不加锁的读.是InnoDB存储引擎下的读取数据的方式(read committed 和 repeatable read).一致性非锁定读,我的理解是它的读取方式是把: 事务隔离级别,MVCC,InnoDB锁结合起来运用到实现Mysql读的一种方式.我们前面提到,mysql读取数据的方式是读MVCC下的快照数据.具体来说,读取mysql数据库时,如果读取的行正在执行DELETE,UPDATE等操作,这时,读取操作不会因此去等待行上的X锁释放,相反,InnoDB会读取行的一个快照数据.这样利用MVCC,InnoDB实现了非阻塞读的实现.极大的提高了数据库的并发性.
但在不同的事务隔离级别下读取的数据的方式也不一样:

(1). 在read committed隔离级别下:
一致性非锁定读总是读取被锁定行的最新一份快照数据. 产生了不可重复读的问题.

(2). 在repeatable read 事务隔离级别下: 一致性非锁定读总是读取事务开始时的行数据版本. 解决不可重复读的问题

4.2 一致性锁定读

还有一种读的方式叫: 一致性锁定读(加锁的读).

1). select .... for update. 加X锁

2). select .... lock in share mode. 加S锁

五: InnoDB 锁的算法

5.1 Record Lock: 但个行记录的锁

5.2 GAP Lock: 间隙锁,锁定一个范围,但不包含记录本身.

5.3 Next-Key Lock: Gap Lock+Record Lock 锁定一个范围并锁定记录本身.

上面所说的锁定的对象均为: 索引记录. 如果InnoDB存储引擎在建立的时候没有设置任何一个索引,那么这时,InnoDB存储引擎会使用隐式的主键进行锁定.
当查询的索引含有唯一属性时,InnoDB存储引擎会对Next-Key Lock进行优化,降级为Record Lock.

下面的2句话是InnoDB在不同隔离级别下产生"不可重复读" 和 "幻读" 和解决它 的根本原因:
InnoDB存储引擎默认的事务隔离级别(repeatable read)下,采用的是 Next-Key Locking的方式来加锁.

read committed隔离级别下采用的是: Record Lock 的方式来加锁.

下面我们来看下 Next-key lock的具体实现:

默认存储引擎下, 比如表A 上的id字段有索引abc, 并且id有 3,8,12,20这几个值,那么该索引可能被Next-key locking区间为:

(负无穷,3)

[3,8)

[8,12),

[12,20),

[20,正无穷)

当事务T1锁定了 [8,12),[12,20)这2个区间时,当插入15时,上面的区间变成:

[8,12),[12,15),[15,20).

但查询索引含有唯一属性时,Next-Key Lock 降级为 Record Lock,仅锁住索引本身.

好,现在表A的id值变成了: 3,8,12,15,20

如果执行下列语句:

select * from A where id>16 for update.

InnoDB会对(16,正无穷) 加锁,

但在 read committed的事务隔离级别下,因为采用Record Lock,只会锁定20这个值.

如果在此时另外一个事务T2,插入了22这个值,此时, read committed 隔离级别下就会产生"幻读"的问题.

但在InnoDB默认存储引擎下的Next-key Lock 模式下,22是插入是会被阻塞的,直到事务T1提交后,释放X锁,才能提交22这值.这样,InnoDB就这样解决了幻读的问题.
现在，我们应该清楚的知道,在不同的事务隔离级别下,mysql InnoDB是如何实现解决 "不可重复读" 和 “幻读” 的问题了吧.

六: 总结:

1, InnoDB用MVCC来实现非阻塞的读操作,不同隔离级别下,MVCC通过读取不同版本的数据来解决"不可重复读" 的问题.

2, InnoDB的默认隔离级别解决2个问题,"不可重复读" 和 "幻读", oracle需要在串行读中解决"幻读"问题. InnoDB的实现方式和一般隔离级别的定义不一致.

3, InnoDB的默认隔离级别采用Next-key Lock(间隙锁) 来解决幻读问题. 而 read committed 隔离级别采用Record锁,因此会产生"幻读"问题.

4, InnoDB的存储引擎不存在锁升级的问题(太多的行锁升级为表锁),来降低锁的开销. 因为不是根据记录来产生行锁的,根据页对锁进行管理.