

锁

基本属性

- 定义

锁机制是用于管理共享资源并发访问的一种机制,保证数据的完整性以及一致性。

- 使用条件

1. 程序存在共享资源,这里的共享资源不仅仅指的是db的一条记录,也可能是内存中的一个变量,一个具体事务的抽象对象等等。
2. 有多个请求同时对资源进行并发访问。

lock锁以及latch锁

lock锁

lock锁的对象是事务,是用来锁定数据库中的对象,如表、页、行;拥有死锁检测机制。

latch锁

latch锁的对象是线程,用户管理同一个进程中不同线程对于内存中临界资源的并发;无死锁检测和处理机制。

latch锁和lock锁的区别

innodb存储引擎书中的图如下:

	lock	latch
对象	事务	线程
保护	数据库内容	内存数据结构
持续时间	整个事务过程	临界资源
模式	行锁、表锁、意向锁	读写锁、互斥量
死锁	通过 waits-for graph、time out 等机制进行死锁检测与处理	无死锁检测与处理机制。仅通过应用程序加锁的顺序（lock leveling）保证无死锁的情况发生
存在于	Lock Manager 的哈希表中	每个数据结构的对象中

InnoDB中的锁

共享锁S

允许事务读一行数据;S锁之间可以重入,也就是说如果事务1已经获取了行r的共享锁另外事务T2也可以获得行r的共享锁。

排他锁X

允许事务删除或者更新一行记录,排他锁和任何锁都不兼容。

X锁和S锁都是作用到数据库上的某一个行的,如果说事务1对某张表的某一行数据加了一个S锁或者X锁,而事务2想要给该表加锁,则需要遍历整张表才知道表中某一行已经被锁住了,这样效率很低,从而mysql引入意向共享

锁IS、意向排他锁IX;数据库在给某一行加S锁前,那么需要给该行所在的表加上IX锁。

意向锁

InnoDB存储引擎支持意向锁的设计比较简练,其意向锁为表级别锁,主要为了在一个事务中揭示下一行即将被请求的锁类型。意向锁是有数据引擎自己维护的,用户无法手动操作意向锁。

意向共享锁(IS Lock)

事务想要获得一张表中某几行的共享锁。

意向排他锁(IX Lock)

事务想要获得一张表中某几行的排他锁

由于InnoDB支持的是行级别的锁,因此意向锁不会阻塞除了全表扫描以外的任何请求。

S锁X锁IS锁IX锁的兼容性

表 6-4 InnoDB 存储引擎中锁的兼容性

	IS	IX	S	X
IS	兼容	兼容	兼容	不兼容
IX	兼容	兼容	不兼容	不兼容
S	兼容	不兼容	兼容	不兼容
X	不兼容	不兼容	不兼容	不兼容

自增锁

表级别自增锁AUTO-INC Locking

- AUTO-INC Locking是MySQL一种特殊的锁,如果表中存在自增字段,MySQL便会自动维护一个自增锁
- AUTO-INC Locking是一个表锁
- AUTO-INC Locking是在一个自增长值插入的SQL语句后立即释放,不是在一个事物完成后才释放,因此提高了并发插入的效率。

新的轻量级互斥量自增长机制

该机制因为没有加表锁等操作,因此大大提高了自增长值插入的性能。但是这种方式在主从复制采用的是sql复制的时候会出现问题。

插入类型

插入类型	说明
insert-like	insert-like 指所有的插入语句，如 INSERT、REPLACE、INSERT...SELECT、REPLACE...SEECT、LOAD DATA 等
simple inserts	simple inserts 指能在插入前就确定插入行数的语句。这些语句包括 INSERT、REPLACE 等。需要注意的是：simple inserts 不包含 INSERT ...ON DUPLICATE KEY UPDATE 这类 SQL 语句
bulk inserts	bulk inserts 指在插入前不能确定得到插入行数的语句，如 INSERT...SELECT、REPLACE...SELECT、LOAD DATA
mixed-mode inserts	mixed-mode inserts 指插入中有一部分的值是自增长的，有一部分是确定的。如 INSERT INTO t1 (c1,c2) VALUES (1,'a'), (NULL,'b'), (5,'c'), (NULL,'d'); 也可以是指 INSERT ...ON DUPLICATE KEY UPDATE 这类 SQL 语句

参数innodb_autoinc_lock_mode

表 6-10 参数 innodb_autoinc_lock_mode 的说明

nnodb_autoinc_lock_mode	说明
0	这是 MySQL5.1.22 版本之前自增长的实现方式，即通过表锁的 AUTO-INC Locking 方式。因为有了新的自增长实现方式，0 这个选项不应该是新版用户的首选项
1	这是该参数的默认值。对于“simple inserts”，该值会用互斥量（mutex）去对内存中的计数器进行累加的操作。对于“bulk inserts”，还是使用传统表锁的 AUTO-INC Locking 方式。在这种配置下，如果不考虑回滚操作，对于自增值列的增长还是连续的。并且在这种方式下，statement-based 方式的 replication 还是能很好地工作。需要注意的是，如果已经使用 AUTO-INC Locing 方式去产生自增长的值，而这时需要再进行“simple inserts”的操作时，还是需要等待 AUTO-INC Locking 的释放
2	在这个模式下，对于所有“INSERT-like”自增长值的产生都是通过互斥量，而不是 AUTO-INC Locking 的方式。显然，这是性能最高的方式。然而，这会带来一定的问题。因为并发插入的存在，在每次插入时，自增长的值可能不是连续的。此外，最重要的是，基于 Statement-Base Replication 会出现问题。因此，使用这个模式，任何时候都应该使用 row-base replication。这样才能保证最大的并发性能及 replication 主从数据的一致

- 当参数值为0的时候,使用表锁的方式进行自增,严格自增不会有问题,但是效率较低。
- 当参数值为1的时候,一次性分配,对分配过程加锁,会导致自增键值非连续,分配策略是当插入N行时,多申请N-1行,则申请的数目为1+N+(N-1)。
- 参数值为2和1相似,但该模式下是来一个分配一个,而不会锁表,只会锁住分配id的过程,和1的区别在于,不会预分配多个,这种方式并发性最高。但是在replication中当binlog_format为statement-based时存在问题。

行锁

Record Lock

当行记录锁,where column = xxx for update

Gap Lock

间隙锁,锁定一个范围,但是不包括记录本身,Where column > xxx for update

Next-Key Lock

Gap锁+Record锁,锁定一个范围,包括记录本身,Where column >= xxx for update

MVCC

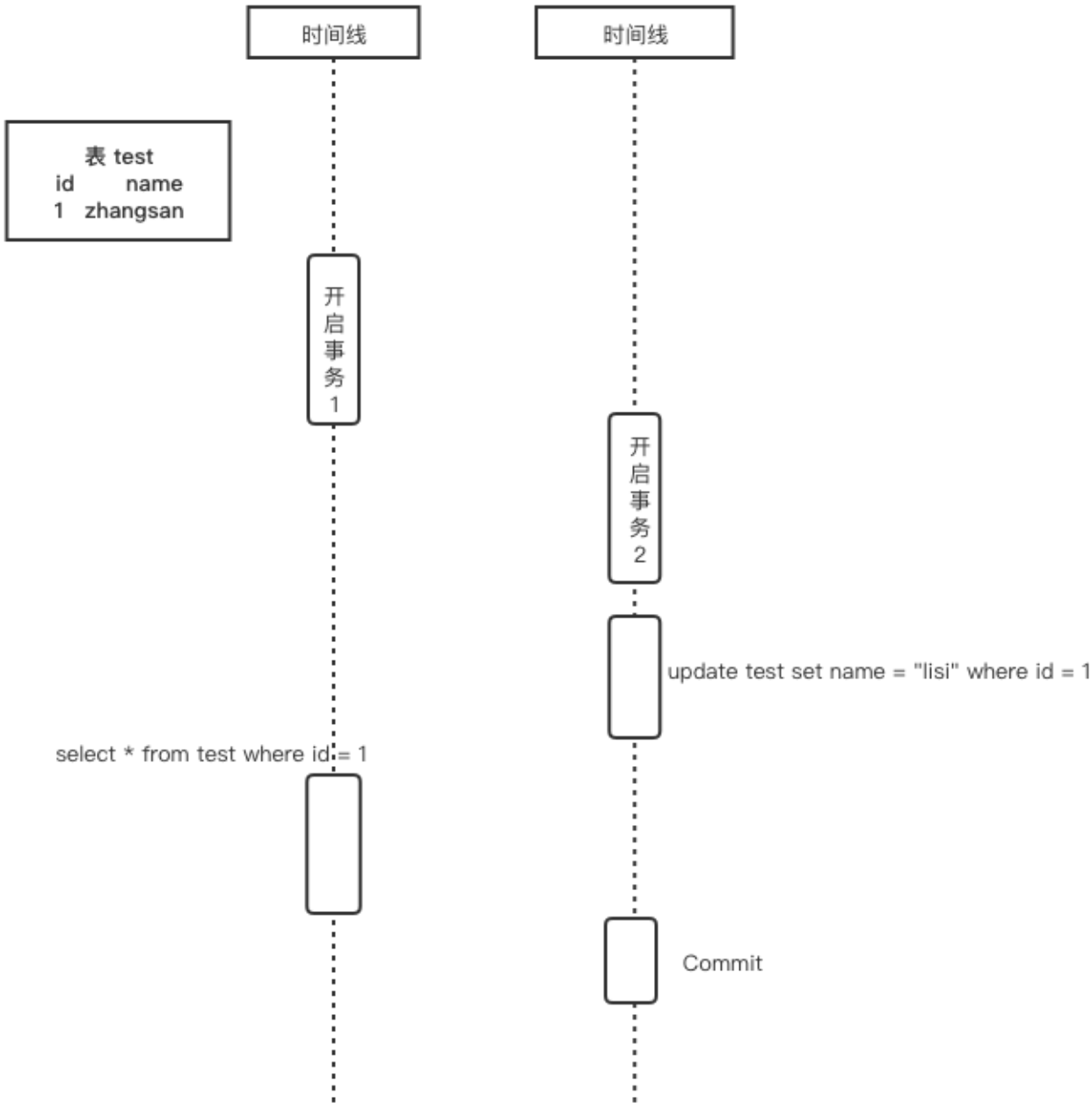
事务与锁

锁问题

脏读


脏读是指一个事务读到了另一个事务未提交的数据,违反了数据库的隔离性。如下图事务1因为事务2没有提交的修改而读不到数据的情况就属于脏读。

事务1	事务2
Begin	Begin
	Update test set name = 'lisi' where id = 1
select * from test where id = 1	
commit	commit



幻读


一个事务读取2次，得到的记录条数不一致。如下图事务1在一个事务内部同一条sql读取出来的条数不一致的情况就属于幻读。Gap Lock 以及 Next-Key Lock可以解决幻读的问题。



事务1	事务2
Begin	Begin
select count(*) from test where id < 10	
	insert into test(id,name)values(2,'lisi')
	commit
select count(*) from test where id < 10	
commit	

不可重复读

一个事务读取2次，得到的结果不一致。如下图事务1在一个事务内部同一条sql读取出来的结果不一致的情况就属于不可重复读。



事务1	事务2
Begin	Begin
select * from test where id =1	
	update test set name = 'lisi' where id = 1
	commit
select * from test where id =1	
commit	

死锁

定义

死锁是指两个或者两个以上的事务在执行过程中因为争夺资源而造成的一种相互等待的现象。

死锁的检测

当前数据库都是使用wait-for graph(等待图)的方式来进行死锁检测。wait-for graph保存着以下两种信息：

-
- 锁的信息链表 事务等待链表

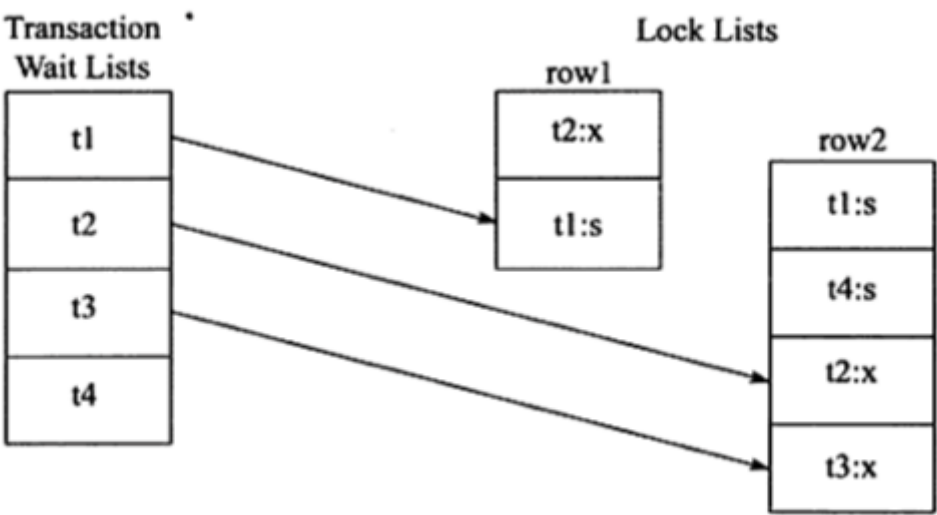


图 6-5 示例事务状态和锁的信息

典型死锁场景

在 Transaction Wait Lists中可以看到共有四个事务t1,t2,t3,t4故在wait-for-graph中应有四个节点。而事务t2对row1占有x锁,事务t1需要等待事务2中的row1的资源,因此在wait-for graph中有条边从节点t1指向节点t2.事务2需要等待事务t1、t4所占的row2对象,故而存在节点t2到节点t1、t4的边。同样,存在节点t3到节点t1、t2、t4的边。

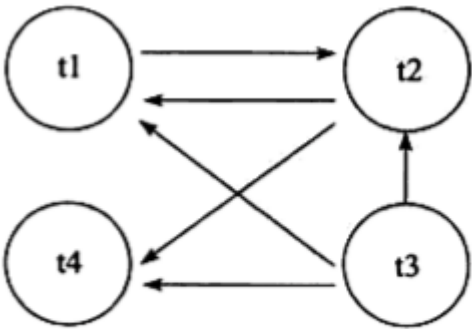


图 6-6 wait-for graph

最终结构如下图：