Mysql 技术内幕：InnoDB 学习（6）

jask

2024-08-11

Mysql 技术内幕

锁

锁是数据库系统区别于文件系统的一个关键特性。锁机制用于管理对共享资源的并发访问。InnoDB 存储引擎会在行级别上对表数据上锁，这固然不错。不过 InnoDB 存储引擎也会在数据库内部其他多个地方使用锁，从而允许对多种不同资源提供并发访问。例如，操作缓冲池中的 LRU 列表，删除、添加、移动 LRU 列表中的元素，为了保证一致性，必须有锁的介入。数据库系统使用锁是为了支持对共享资源进行并发访问，提供数据的完整性和一致性。

Lock 和 Latch

latch 一般称为闩锁（轻量级的锁），因为其要求锁定的时间必须非常短。若持续的时间长，则应用的性能会非常差。在 InnoDB 存储引擎中，latch 又可以分为 mutex（互斥量）和 rwlock（读写锁）。其目的是用来保证并发线程操作临界资源的正确性，并且通常没有死锁检测的机制。

lock 的对象是事务，用来锁定的是数据库中的对象，如表、页、行。并且一般 lock 的对象仅在事务 commit 或 rollback 后进行释放（不同事务隔离级别释放的时间可能不同）。此外，lock，正如在大多数数据库中一样，是有死锁机制的。表 6-1 显示了 lock 与 latch 的不同。

表 6-1　lock 与 latch 的比较

	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

Figure 1: 二者的比较

锁的类型

InnoDB 存储引擎实现了如下两种标准的行级锁：

共享锁（S Lock），允许事务读一行数据。

排他锁（X Lock），允许事务删除或更新一行数据。

表 6-3 排他锁和共享锁的兼容性

	X	S
X	不兼容	不兼容
S	不兼容	兼容

Figure 2: 行锁排他锁共享锁兼容性

此外，InnoDB 存储引擎支持多粒度（granular）锁定，这种锁定允许事务在行级上的锁和表级上的锁同时存在。为了支持在不同粒度上进行加锁操作，InnoDB 存储引擎支持一种额外的锁方式，称之为意向锁（Intention Lock）。意向锁是将锁定的对象分为多个层次，意向锁意味着事务希望在更细粒度（fine granularity）上进行加锁，如图所示。

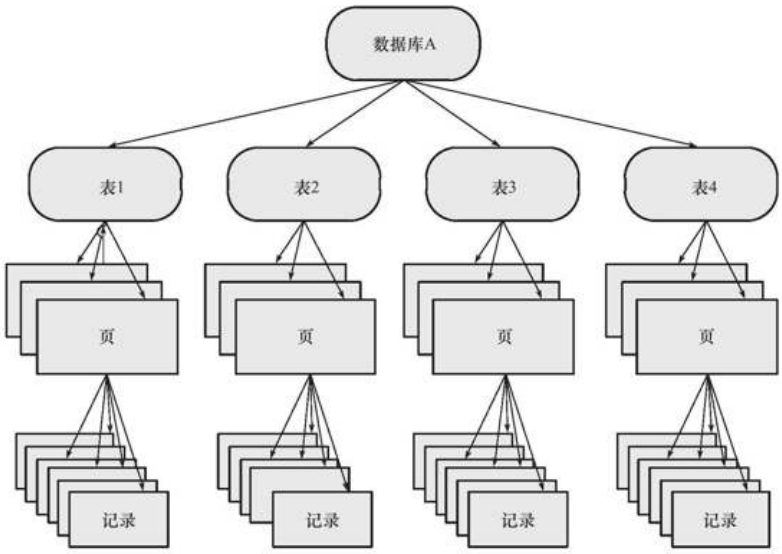

图 6-3 层次结构

Figure 3: 层次结构

如果需要对页上的记录 r 进行上 X 锁，那么分别需要对数据库 A、表、页上意向锁 IX，最后对记录 r 上 X 锁。若其中任何一个部分导致等待，那么该操作需要等待粗粒度锁的完成。举例来说，在对记录 r 加 X 锁之前，已经有事务对表 1 进行了 S 表锁，那么表 1 上已

存在 S 锁，之后事务需要对记录 r 在表 1 上加上 IX，由于不兼容，所以该事务需要等待表锁操作的完成。

InnoDB 存储引擎支持意向锁设计比较简练，其意向锁即为表级别的锁。设计目的主要是为了在一个事务中揭示下一行将被请求的锁类型。其支持两种意向锁：

1）意向共享锁（IS Lock），事务想要获得一张表中某几行的共享锁

2）意向排他锁（IX Lock），事务想要获得一张表中某几行的排他锁

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

Figure 4: 锁的兼容性

一致非锁定读 一致性的非锁定读（consistent nonlocking read）是指 InnoDB 存储引擎通过行多版本控制（multi versioning）的方式来读取当前执行时间数据库中行的数据。如果读取的行正在执行 DELETE 或 UPDATE 操作，这时读取操作不会因此去等待行上锁的释放。相反地，InnoDB 存储引擎会去读取行的一个快照数据。

之所以称其为非锁定读，因为不需要等待访问的行上 X 锁的释放。快照数据是指该行的之前版本的数据，该实现是通过 undo 段来完成。而 undo 用来在事务中回滚数据，因此快照数据本身是没有额外的开销。此外，读取快照数据是不需要上锁的，因为没有事务需要对历史的数据进行修改操作。

在事务隔离级别 READ COMMITTED 和 REPEATABLE READ（InnoDB 存储引擎的默认事务隔离级别）下，InnoDB 存储引擎使用非锁定的一致性读。然而，对于快照数据的定义却不相同。在 READ COMMITTED 事务隔离级别下，对于快照数据，非一致性读总是读取被锁定行的最新一份快照数据。而在 REPEATABLE READ 事务隔离级别下，对于快照数据，非一致性读总是读取事务开始时的行数据版本。

一致锁定读 SELECT…FOR UPDATE

SELECT…LOCK IN SHARE MODE

SELECT…FOR UPDATE 对读取的行记录加一个 X 锁，其他事务不能对已锁定的行加上任何锁。SELECT…LOCK IN SHARE MODE 对读取的行记录加一个 S 锁，其他事务可以向被锁定的行加 S 锁，但是如果加 X 锁，则会被阻塞。

对于一致性非锁定读，即使读取的行已被执行了 SELECT…FOR UPDATE，也是可以进行读取的，这和之前讨论的情况一样。此外，SELECT…FOR UPDATE，SELECT…LOCK IN SHARE MODE 必须在一个事务中，当事务提交了，锁也就释放了。因此在使用上述两句 SELECT 锁定语句时，务必加上 BEGIN，START TRANSACTION 或者 SET AUTOCOMMIT=0。

自增长与锁 自增长在数据库中是非常常见的一种属性，也是很多 DBA 或开发人员首选的主键方式。在 InnoDB 存储引擎的内存结构中，对每个含有自增长值的表都有一个自增长计数器（auto-increment counter）。当对含有自增长的计数器的表进行插入操作时，这个计数器会被初始化，执行如下的语句来得到计数器的值：

```
SELECT MAX(auto_inc_col) FROM t FOR UPDATE;
```

插入操作会依据这个自增长的计数器值加 1 赋予自增长列。这个实现方式称做 AUTO-INC Locking。这种锁其实是采用一种特殊的表锁机制，为了提高插入的性能，锁不是在一个事务完成后才释放，而是在完成对自增长值插入的 SQL 语句后立即释放。

外键和锁 在 InnoDB 存储引擎中，对于一个外键列，如果没有显式地对这个列加索引，InnoDB 存储引擎自动对其加一个索引，因为这样可以避免表锁——这比 Oracle 数据库做得好，Oracle 数据库不会自动添加索引，用户必须自己手动添加，这也导致了 Oracle 数据库中可能产生死锁。

对于外键值的插入或更新，首先需要查询父表中的记录，即 SELECT 父表。但是对于父表的 SELECT 操作，不是使用一致性非锁定读的方式，因为这样会发生数据不一致的问题，因此这时使用的是 SELECT…LOCK IN SHARE MODE 方式，即主动对父表加一个 S 锁。如果这时父表上已经这样加 X 锁，子表上的操作会被阻塞。

行锁的 3 种算法 InnoDB 存储引擎有 3 种行锁的算法，其分别是：

Record Lock：单个行记录上的锁

Gap Lock：间隙锁，锁定一个范围，但不包含记录本身

Next-Key Lock：Gap Lock+Record Lock，锁定一个范围，并且锁定记录本身

Record Lock 总是会去锁住索引记录，如果 InnoDB 存储引擎表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来进行锁定。

Next-Key Lock 是结合了 Gap Lock 和 Record Lock 的一种锁定算法，在 Next-Key Lock 算法下，InnoDB 对于行的查询都是采用这种锁定算法。例如一个索引有 10，11，13 和 20 这四个值，那么该索引可能被 Next-Key Locking 的区间为：

采用 Next-Key Lock 的锁定技术称为 Next-Key Locking。其设计的目的是为了解决 Phantom Problem，这将在下一小节中介绍。而利用这种锁定技术，锁定的不是单个值，而是一个范围，是谓词锁（predict lock）的一种改进。除了 next-key locking，还有 previous-key locking 技术。同样上述的索引 10、11、13 和 20，若采用 previous-key locking 技术，那么可锁定的区间为：

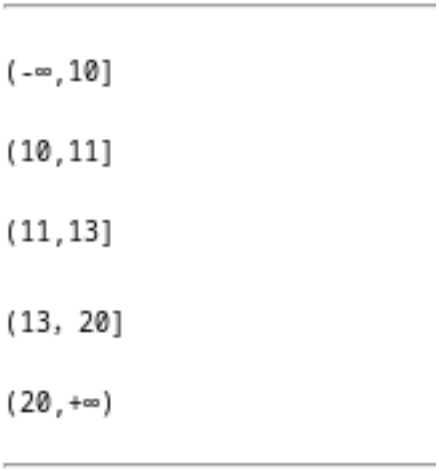

Figure 5: 可能区间

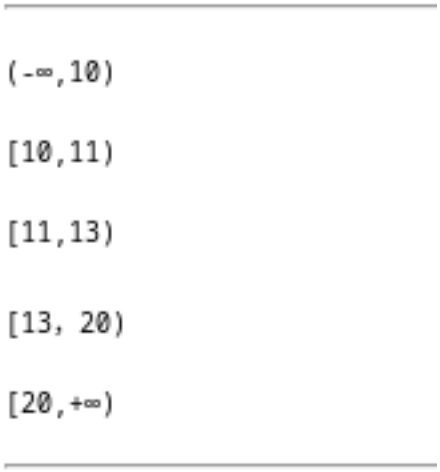

Figure 6: 可能区间

若事务T1已经通过next-key locking锁定了如下范围：

(10,11]、(11，13]

当插入新的记录12时，则锁定的范围会变成：

(10,11]、(11,12]、(12，13]

用户可以通过以下两种方式来显式地关闭 Gap Lock：

将事务的隔离级别设置为 READ COMMITTED

将参数 innodb_locks_unsafe_for_binlog 设置为 1

解决 Phantom Problem 在默认的事务隔离级别下，即 REPEATABLE READ 下，InnoDB 存储引擎采用 Next-Key Locking 机制来避免 Phantom Problem（幻像问题）。这点可能不同于与其他的数据库，如 Oracle 数据库，因为其可能需要在 SERIALIZABLE 的事务隔离级别下才能解决 Phantom Problem。

Phantom Problem 是指在同一事务下，连续执行两次同样的 SQL 语句可能导致不同的结果，第二次的 SQL 语句可能会返回之前不存在的行。

InnoDB 存储引擎采用 Next-Key Locking 的算法避免 Phantom Problem。对于上述的 SQL 语句 SELECT*FROM t WHERE a > 2 FOR UPDATE，其锁住的不是 5 这单个值，而是对（2，INF）这个范围加了 X 锁。因此任何对于这个范围的插入都是不被允许的，从而避免 Phantom Problem。

InnoDB 存储引擎默认的事务隔离级别是 REPEATABLE READ，在该隔离级别下，其采用 Next-Key Locking 的方式来加锁。而在事务隔离级别 READ COMMITTED 下，其仅采用 Record Lock，因此在上述的示例中，会话 A 需要将事务的隔离级别设置为 READ COMMITTED。

锁问题

通过锁定机制可以实现事务的隔离性要求，使得事务可以并发地工作。锁提高了并发，但是却会带来潜在的问题。不过好在因为事务隔离性的要求，锁只会带来三种问题，如果可以防止这三种情况的发生，那将不会产生并发异常。

脏读 在理解脏读（Dirty Read）之前，需要理解脏数据的概念。但是脏数据和之前所介绍的脏页完全是两种不同的概念。脏页指的是在缓冲池中已经被修改的页，但是还没有刷新到磁盘中，即数据库实例内存中的页和磁盘中的页的数据是不一致的，当然在刷新到磁盘之前，日志都已经被写入到了重做日志文件中。而所谓脏数据是指事务对缓冲池中行记录的修改，并且还没有被提交（commit）。

对于脏页的读取，是非常正常的。脏页是因为数据库实例内存和磁盘的异步造成的，这并不影响数据的一致性（或者说两者最终会达到一致性，即当脏页都刷回到磁盘）。并且因为脏页的刷新是异步的，不影响数据库的可用性，因此可以带来性能的提高。

脏数据却截然不同，脏数据是指未提交的数据，如果读到了脏数据，即一个事务可以读到另外一个事务中未提交的数据，则显然违反了数据库的隔离性。

脏读指的就是在不同的事务下，当前事务可以读到另外事务未提交的数据，简单来说就是可以读到脏数据。

不可重复读 不可重复读是指在一个事务内多次读取同一数据集合。在这个事务还没有结束时，另外一个事务也访问该同一数据集合，并做了一些 DML 操作。因此，在第一个事务中的两次读数据之间，由于第二个事务的修改，那么第一个事务两次读到的数据可能是不一样的。这样就发生了在一个事务内两次读到的数据是不一样的情况，这种情况称为不可重复读。

不可重复读和脏读的区别是：脏读是读到未提交的数据，而不可重复读读到的却是已经提交的数据，但是其违反了数据库事务一致性的要求。

一般来说，不可重复读的问题是可以接受的，因为其读到的是已经提交的数据，本身并不会带来很大的问题。因此，很多数据库厂商（如 Oracle、Microsoft SQL Server）将其数据库事务的默认隔离级别设置为 READ COMMITTED，在这种隔离级别下允许不可重复读的现象。

在 InnoDB 存储引擎中，通过使用 Next-Key Lock 算法来避免不可重复读的问题。在 MySQL 官方文档中将不可重复读的问题定义为 Phantom Problem，即幻像问题。在 Next-Key Lock 算法下，对于索引的扫描，不仅是锁住扫描到的索引，而且还锁住这些索引覆盖的范围（gap）。因此在这个范围内的插入都是不允许的。这样就避免了另外的事务在这个范围内插入数据导致的不可重复读的问题。因此，InnoDB 存储引擎的默认事务隔离级别是 READ REPEATABLE，采用 Next-Key Lock 算法，避免了不可重复读的现象。

丢失更新 丢失更新是另一个锁导致的问题，简单来说其就是一个事务的更新操作会被另一个事务的更新操作所覆盖，从而导致数据的不一致。

1）事务 T1 将行记录 r 更新为 v1，但是事务 T1 并未提交。

2）与此同时，事务 T2 将行记录 r 更新为 v2，事务 T2 未提交。

3）事务 T1 提交。

4）事务 T2 提交。

但是，在当前数据库的任何隔离级别下，都不会导致数据库理论意义上的丢失更新问题。这是因为，即使是 READ UNCOMMITTED 的事务隔离级别，对于行的 DML 操作，需要对行或其他粗粒度级别的对象加锁。因此在上述步骤 2）中，事务 T2 并不能对行记录 r 进行更新操作，其会被阻塞，直到事务 T1 提交。

阻塞

因为不同锁之间的兼容性关系，在有些时刻一个事务中的锁需要等待另一个事务中的锁释放它所占用的资源，这就是阻塞。阻塞并不是一件坏事，其是为了确保事务可以并发且正常地运行。

死锁

解决死锁问题最简单的一种方法是超时，即当两个事务互相等待时，当一个等待时间超过设置的某一阈值时，其中一个事务进行回滚，另一个等待的事务就能继续进行。在 InnoDB 存储引擎中，参数 innodb_lock_wait_timeout 用来设置超时的时间。

锁升级

锁升级（Lock Escalation）是指将当前锁的粒度降低。举例来说，数据库可以把一个表的 1000 个行锁升级为一个页锁，或者将页锁升级为表锁。如果在数据库的设计中认为锁是一种稀有资源，而且想避免锁的开销，那数据库中会频繁出现锁升级现象。