1. MySql锁机制
   1. 概述

### 1.1.1 定义

锁是计算机协调多个进程或线程并发访问某一资源的机制。

在数据库中，除传统的计算资源（如CPU、RAM、I/O等）的争用以外，数据也是一种供许多用户共享的资源。如何保证数据并发访问的一致性、有效性是所有数据库必须解决的一个问题，锁冲突也是影响数据库并发访问性能的一个重要因素。从这个角度来说，锁对数据库而言显得尤其重要，也更加复杂。

1.1.2 生活购物

打个比方，我们到淘宝上买一件商品，商品只有一件库存，这个时候如果还有另一个人买，

那么如何解决是你买到还是另一个人买到的问题？



这里肯定要用到事务，我们先从库存表中取出物品数量，然后插入订单，付款后插入付款表信息，

然后更新商品数量。在这个过程中，使用锁可以对有限的资源进行保护，解决隔离和并发的矛盾。

1.1.3锁的分类

从对数据操作的类型分

读锁(共享锁)：针对同一份数据，多个读操作可以同时进行而不会互相影响。

写锁（排它锁）：当前写操作没有完成前，它会阻断其他写锁和读锁。

从对数据操作的粒度分

为了尽可能提高数据库的并发度，每次锁定的数据范围越小越好，理论上每次只锁定当前操作的数据的方案会得到最大的并发度，但是管理锁是很耗资源的事情（涉及获取，检查，释放锁等动作），因此数据库系统需要在高并发响应和系统性能两方面进行平衡，这样就产生了“锁粒度（Lock granularity）”的概念。

一种提高共享资源并发发性的方式是让锁定对象更有选择性。尽量只锁定需要修改的部分数据，而不是所有的资源。更理想的方式是，只对会修改的数据片进行精确的锁定。任何时候，在给定的资源上，锁定的数据量越少，则系统的并发程度越高，只要相互之间不发生冲突即可。

表锁

行锁

* 1. 三锁
     1. 表锁(偏读)

特点

偏向MyISAM存储引擎，开销小，加锁快；无死锁；锁定粒度大，发生锁冲突的概率最高, 并发度最低。

案例分析

建表SQL

【表级锁分析--建表SQL】

create table mylock(

id int not null primary key auto\_increment,

name varchar(20)

)engine myisam;

insert into mylock(name) values('a');

insert into mylock(name) values('b');

insert into mylock(name) values('c');

insert into mylock(name) values('d');

insert into mylock(name) values('e');

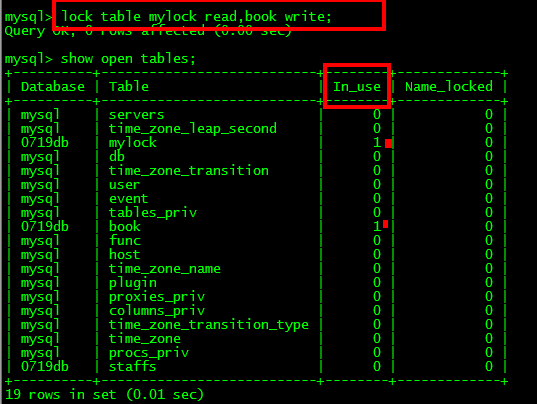
select \* from mylock;

【手动增加表锁】

lock table 表名字1 read(write)，表名字2 read(write)，其它;

【查看表上加过的锁】

show open tables;



【释放表锁】

unlock tables;

加读锁

我们为mylock表加read锁(读阻塞写例子)

|  |  |
| --- | --- |
| session\_1 | session\_2 |
| 获得表mylock的READ锁定  graphic | 连接终端 |
| 当前session可以查询该表记录  graphic | 其他session也可以查询该表的记录  graphic |
| 当前session不能查询其它没有锁定的表  graphic | 其他session可以查询或者更新未锁定的表  graphic |
| 当前session中插入或者更新锁定的表都会提示错误：  graphic | 其他session插入或者更新锁定表**会一直等待**获得锁：  graphic |
| 释放锁  graphic | Session2获得锁，插入操作完成：  graphic |
|  |  |
|  |  |

加写锁

mylockwrite(MyISAM)

|  |  |
| --- | --- |
| session\_1 | session\_2 |
| 获得表mylock的WRITE锁定  graphic | 待Session1开启写锁后，session2再连接终端 |
| 当前session对锁定表的查询+更新+插入操作都可以执行：  graphic | 其他session对锁定表的查询被阻塞，需要等待锁被释放：  graphic  在锁表前，如果session2有数据缓存，锁表以后，在锁住的表不发生改变的情况下session2可以读出缓存数据，一旦数据发生改变，缓存将失效，操作将被阻塞住。 |
| 释放锁  graphic | Session2获得锁，查询返回：  graphic |
|  |  |
|  |  |
|  |  |
|  |  |

结论

MyISAM在执行查询语句（SELECT）前，会自动给涉及的所有表加读锁，在执行增删改操作前，会自动给涉及的表加写锁。

MySQL的表级锁有两种模式：

表共享读锁（Table Read Lock）

表独占写锁（Table Write Lock）

|  |  |  |
| --- | --- | --- |
| 锁类型 | 他人可读 | 他人可写 |
| 读锁 | 是 | 否 |
| 写锁 | 否 | 否 |

结论：

结合上表，所以对MyISAM表进行操作，会有以下情况：

1、对MyISAM表的读操作（加读锁），不会阻塞其他进程对同一表的读请求，但会阻塞对同一表的写请求。只有当读锁释放后，才会执行其它进程的写操作。

2、对MyISAM表的写操作（加写锁），会阻塞其他进程对同一表的读和写操作，只有当写锁释放后，才会执行其它进程的读写操作。

简而言之，就是读锁会阻塞写，但是不会堵塞读。而写锁则会把读和写都堵塞

* + 1. 行锁(偏写)

特点

偏向InnoDB存储引擎，开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低,并发度也最高。

InnoDB与MyISAM的最大不同有两点：一是支持事务（TRANSACTION）；二是采用了行级锁

由于行锁支持事务，复习老知识

事务（Transaction）及其ACID属性

事务是由一组SQL语句组成的逻辑处理单元，事务具有以下4个属性，通常简称为事务的ACID属性。

l 原子性（Atomicity）：事务是一个原子操作单元，其对数据的修改，要么全都执行，要么全都不执行。

l 一致性（Consistent）：在事务开始和完成时，数据都必须保持一致状态。这意味着所有相关的数据规则都必须应用于事务的修改，以保持数据的完整性；事务结束时，所有的内部数据结构（如B树索引或双向链表）也都必须是正确的。

l 隔离性（Isolation）：数据库系统提供一定的隔离机制，保证事务在不受外部并发操作影响的“独立”环境执行。这意味着事务处理过程中的中间状态对外部是不可见的，反之亦然。

l 持久性（Durable）：事务完成之后，它对于数据的修改是永久性的，即使出现系统故障也能够保持。

并发事务处理带来的问题

更新丢失(Lost Update)

当两个或多个事务选择同一行，然后基于最初选定的值更新该行时，由于每个事务都不知道其他事务的存在，就会发生丢失更新问题－－最后的更新覆盖了由其他事务所做的更新。

例如，两个程序员修改同一java文件。每程序员独立地更改其副本，然后保存更改后的副本，这样就覆盖了原始文档。最后保存其更改副本的编辑人员覆盖前一个程序员所做的更改。

如果在一个程序员完成并提交事务之前，另一个程序员不能访问同一文件，则可避免此问题。

脏读(Dirty Reads)

一个事务正在对一条记录做修改，在这个事务完成并提交前，这条记录的数据就处于不一致状态；这时，另一个事务也来读取同一条记录，如果不加控制，第二个事务读取了这些“脏”数据，并据此做进一步的处理，就会产生未提交的数据依赖关系。这种现象被形象地叫做”脏读”。

一句话：事务A读取到了事务B已修改但尚未提交的的数据，还在这个数据基础上做了操作。此时，如果B事务回滚，A读取

的数据无效，不符合一致性要求。

不可重复读(Non-Repeatable Reads)

在一个事务内，多次读同一个数据。在这个事务还没有结束时，另一个事务也访问该同一数据。那么，在第一个事务的两次读数据之间。由于第二个事务的修改，那么第一个事务读到的数据可能不一样，这样就发生了在一个事务内两次读到的数据是不一样的，因此称为不可重复读，即原始读取不可重复。

一句话：一个事务范围内两个相同的查询却返回了不同数据。

幻读(Phantom Reads)

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”。

一句话：事务A 读取到了事务B提交的新增数据，不符合隔离性。

事务隔离级别

脏读”、“不可重复读”和“幻读”，其实都是数据库读一致性问题，必须由数据库提供一定的事务隔离机制来解决。



数据库的事务隔离越严格，并发副作用越小，但付出的代价也就越大，因为事务隔离实质上就是使事务在一定程度上 “串行化”进行，这显然与“并发”是矛盾的。同时，不同的应用对读一致性和事务隔离程度的要求也是不同的，比如许多应用对“不可重复读”和“幻读”并不敏感，可能更关心数据并发访问的能力。

常看当前数据库的事务隔离级别：show variables like 'tx\_isolation';

案例分析

建表SQL

create table test\_innodb\_lock (a int(11),b varchar(16))engine=innodb;

insert into test\_innodb\_lock values(1,'b2');

insert into test\_innodb\_lock values(3,'3');

insert into test\_innodb\_lock values(4,'4000');

insert into test\_innodb\_lock values(5,'5000');

insert into test\_innodb\_lock values(6,'6000');

insert into test\_innodb\_lock values(7,'7000');

insert into test\_innodb\_lock values(8,'8000');

insert into test\_innodb\_lock values(9,'9000');

insert into test\_innodb\_lock values(1,'b1');

create index test\_innodb\_a\_ind on test\_innodb\_lock(a);

create index test\_innodb\_lock\_b\_ind on test\_innodb\_lock(b);

select \* from test\_innodb\_lock;

行锁定基本演示

行锁定基本演示

|  |  |
| --- | --- |
| Session\_1 | Session\_2 |
| graphic | graphic |
| 更新但是不提交，没有手写commit;  graphic | Session\_2被阻塞，只能等待  graphic |
| 提交更新  graphic | 解除阻塞，更新正常进行  graphic |
|  | commit命令执行 |
| 下面试试1号会话更新a =1 | 下面试试2号会话更新a =9 |

无索引行锁升级为表锁

|  |  |
| --- | --- |
| Session\_1 | Session\_2 |
| 正常情况，各自锁定各自的行，互相不影响，一个2000另一个3000 |  |
| graphic | graphic |
| 由于在column字段b上面建了索引，如果没有正常使用，会导致行锁变表锁 |  |
| 比如没加单引号导致索引失效，行锁变表锁  graphic | 被阻塞，等待。只到Session\_1提交后才阻塞解除，完成更新  graphic |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Select也可以加锁

读锁

select ..lock in share mode

共享锁(Share Lock)

共享锁又称读锁，是读取操作创建的锁。其他用户可以并发读取数据，但任何事务都不能对数据进行修改（获取数据上的排他锁），直到已释放所有共享锁。

如果事务T对数据A加上共享锁后，则其他事务只能对A再加共享锁，不能加排他锁。获准共享锁的事务只能读数据，不能修改数据。

用法

SELECT ... LOCK IN SHARE MODE;

在查询语句后面增加 LOCK IN SHARE MODE ，Mysql会对***查询结果中的每行都加共享锁，***当没有其他线程对查询结果集中的任何一行使用排他锁时，可以成功申请共享锁，否则会被阻塞。其他线程也可以读取使用了共享锁的表（行？），而且这些线程读取的是同一个版本的数据。

写锁

select... for update

排他锁（eXclusive Lock）

排他锁又称写锁，如果事务T对数据A加上排他锁后，则其他事务不能再对A加任任何类型的封锁。获准排他锁的事务既能读数据，又能修改数据。

用法

SELECT ... FOR UPDATE;

在查询语句后面增加 FOR UPDATE ，Mysql会对查询结果中的每行都加排他锁，当没有其他线程对查询结果集中的任何一行使用排他锁时，可以成功申请排他锁，否则会被阻塞。

间隙锁危害

间隙锁带来的插入问题

|  |  |
| --- | --- |
| Session\_1 | Session\_2 |
| graphic | 阻塞产生，暂时不能插入  graphic |
| commit; | 阻塞解除，完成插入  graphic |
|  |  |

【什么是间隙锁】

当我们用范围条件而不是相等条件检索数据，并请求共享或排他锁时，InnoDB会给符合条件的已有数据记录的索引项加锁；对于键值在条件范围内但并不存在的记录，叫做“间隙（GAP)”，

InnoDB也会对这个“间隙”加锁，这种锁机制就是所谓的间隙锁（GAP Lock）。

【危害】

因为Query执行过程中通过过范围查找的话，他会锁定整个范围内所有的索引键值，即使这个键值并不存在。

间隙锁有一个比较致命的弱点，就是当锁定一个范围键值之后，即使某些不存在的键值也会被无辜的锁定，而造成在锁定的时候无法插入锁定键值范围内的任何数据。在某些场景下这可能会对性能造成很大的危害

案列结论

Innodb存储引擎由于实现了行级锁定，虽然在锁定机制的实现方面所带来的性能损耗可能比表级锁定会要更高一些，但是在整体并发处理能力方面要远远优于MyISAM的表级锁定的。当系统并发量较高的时候，Innodb的整体性能和MyISAM相比就会有比较明显的优势了。

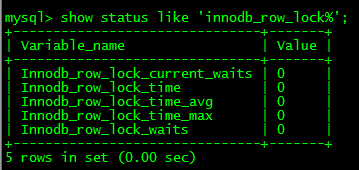
但是，Innodb的行级锁定同样也有其脆弱的一面，当我们使用不当的时候，可能会让Innodb的整体性能表现不仅不能比MyISAM高，甚至可能会更差。

行锁分析

【如何分析行锁定】

通过检查InnoDB\_row\_lock状态变量来分析系统上的行锁的争夺情况

mysql>show status like 'innodb\_row\_lock%';



对各个状态量的说明如下：

Innodb\_row\_lock\_current\_waits：当前正在等待锁定的数量；

Innodb\_row\_lock\_time：从系统启动到现在锁定总时间长度；

Innodb\_row\_lock\_time\_avg：每次等待所花平均时间；

Innodb\_row\_lock\_time\_max：从系统启动到现在等待最常的一次所花的时间；

Innodb\_row\_lock\_waits：系统启动后到现在总共等待的次数；

对于这5个状态变量，比较重要的主要是

Innodb\_row\_lock\_time\_avg（等待平均时长），

Innodb\_row\_lock\_waits（等待总次数）

Innodb\_row\_lock\_time（等待总时长）这三项。

尤其是当等待次数很高，而且每次等待时长也不小的时候，我们就需要分析系统中为什么会有如此多的等待，然后根据分析结果着手指定优化计划。

最后可以通过

SELECT \* FROM information\_schema.INNODB\_TRX\G;

来查询正在被锁阻塞的sql语句。

优化建议

尽可能让所有数据检索都通过索引来完成，避免无索引行锁升级为表锁。

尽可能较少检索条件，避免间隙锁

尽量控制事务大小，减少锁定资源量和时间长度

锁住某行后，尽量不要去调别的行或表，赶紧处理被锁住的行然后释放掉锁。

涉及相同表的事务，对于调用表的顺序尽量保持一致。

在业务环境允许的情况下,尽可能低级别事务隔离

* + 1. 页锁

开销和加锁时间界于表锁和行锁之间；会出现死锁；锁定粒度界于表锁和行锁之间，并发度一般。

了解一下即可