

您的位置: 首页 → 数据库 → Mysql → Mysql 行锁、表锁、死锁

请输入关键词

Mysql锁机制之行锁、表锁、死锁的实现

更新时间: 2022年03月16日 09:24:03 作者: 这是王姑娘的微博

本文主要介绍了Mysql锁机制之行锁、表锁、死锁的实现，文中通过示例代码介绍的非常详细，具有一定的参考价值，感兴趣的小伙伴们可以参考一下

目录

- 一、Mysql锁是什么？锁有哪些类别？
- 二、行锁和表锁的区别
- 三、InnoDB死锁概念和死锁案例
 - 死锁场景一之select for update:
 - 死锁场景二之两个update
- 四、程序开发过程中应该如何注意避免死锁

一、Mysql锁是什么？锁有哪些类别？

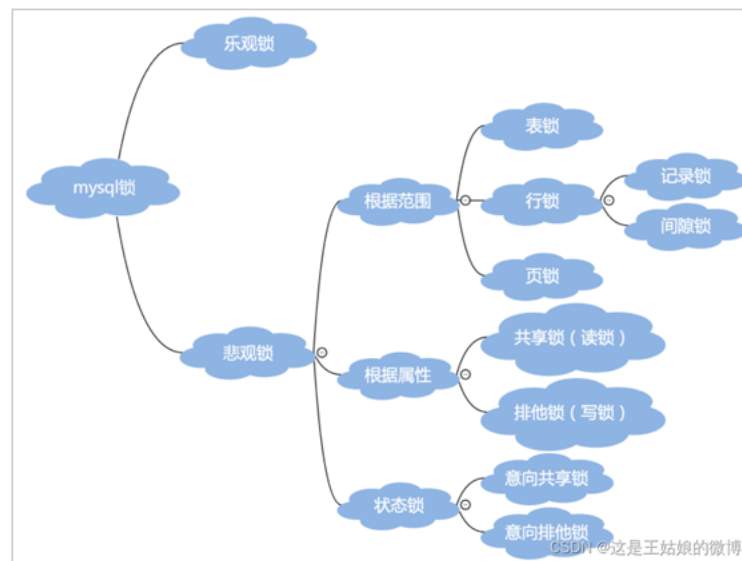
锁定义:

同一时间同一资源只能被一个线程访问

在数据库中，除传统的计算资源（如CPU、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。

乐观锁用的最多的就是数据的版本记录来体现 version，其实就是一个标识。

例如: `update test set a=a-1 where id=100 and a> 0` ; 对应的version就是a字段，并不一定非得要求有一个字段叫做version，要求的是有这个字段，同时当满足这个条件的时候才会触发



锁的分类:

从对数据操作的类型分法 (读或写)

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法

表级锁: 表级锁是MySQL中锁定粒度最大的一种锁, 表示对当前操作的整张表加锁(MyISAM引擎默认表级锁, 也只支持表级锁)。比如说更新一张10万表数据中的一条数据, 在这条update没提交事务之前, 其它事务是会被排斥掉的, 粒度很大。

行级锁: 行级锁是Mysql中锁定粒度最细的一种锁, 表示只针对当前操作的行进行加锁(基于索引实现的, 所以一旦某个加锁操作没有使用索引, 那么该锁就会退化为表锁)

页级锁: 页级锁是MySQL中锁定粒度介于行级锁和表级锁中间的一种锁, 一次锁定相邻的一组记录

从并发角度的分发--实际上乐观锁和悲观锁只是一种思想

悲观锁: 对数据被外界 (包括本系统当前的其他事务, 以及来自外部系统的事务处理) 修改持保守态度(悲观), 因此, 在整个数据处理过程中, 将数据处于锁定状态。

乐观锁: 乐观锁假设认为数据一般情况下不会造成冲突, 所以在数据进行提交更新的时候, 才会正式对数据的冲突与否进行检测, 如果发现冲突了, 则让返回错误信息再进行业务重试

其他锁：

间隙锁：在条件查询中，如：where id>100，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）”，间隙的目的是为了防止幻读

意向锁：意向锁分为 intention shared lock (IS) 和 intention exclusive lock (IX)，意向锁的目的就是表明有事务正在或者将要锁住某个表中的行

二、行锁和表锁的区别

表级锁是MySQL中锁定粒度最大的一种锁，表示对当前操作的整张表加锁，它实现简单。最常使用的MYISAM与INNODB都支持表级锁定。

特点：**开销小，加锁快**；不会出现死锁；锁定粒度大，发出锁冲突的概率最高，并发度最低。

行级锁是Mysql中锁定粒度最细的一种锁，表示只针对当前操作的行进行加锁。行级锁能大大减少数据库操作的冲突。其加锁粒度最小，但加锁的开销也最大。

特点：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

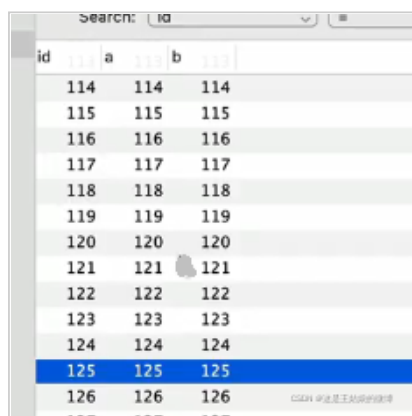
使用：InnoDB行锁是通过给索引上的索引项加锁来实现的，只有**通过索引条件检索数据**，InnoDB才使用行级锁，否则，InnoDB将使用表锁

下面这个update语句，b是一般字段不是索引列的话，那么此时行级锁将改为表级锁。

```
1 | update from test set a=100 where b='100';
```

现在举个实际例子操作一下，看看innodb是怎么来用行锁的。

当前表中数据：



| id | a | b |
|-----|-----|-----|
| 114 | 114 | 114 |
| 115 | 115 | 115 |
| 116 | 116 | 116 |
| 117 | 117 | 117 |
| 118 | 118 | 118 |
| 119 | 119 | 119 |
| 120 | 120 | 120 |
| 121 | 121 | 121 |
| 122 | 122 | 122 |
| 123 | 123 | 123 |
| 124 | 124 | 124 |
| 125 | 125 | 125 |
| 126 | 126 | 126 |

首先开启两个session会话窗口，然后将mysql事务级别设置成不提交级别：

会话一窗口:

```
4 set autocommit = 0  第一步
5 select * from test where id =125;  第二步查询一下
6 update test set b=200 where id =125;  第三步执行一下
7
8
```

这是在会话1当中按顺序执行，此时事务并没有提交，所以这行125是被锁住的。

| id | a | b |
|-----|-----|-----|
| 125 | 125 | 125 |

CSDN @这是王姑娘的微博

会话二窗口:

```
5
6
7 select * from test where id =125;  第一步查询，结果和会话一的查询结果一样。
8 update test set b=200 where id =125;  第二步执行这个update语句，出现了
9                                     下面的弹窗
10
```

这是会话2



CSDN @这是王姑娘的微博

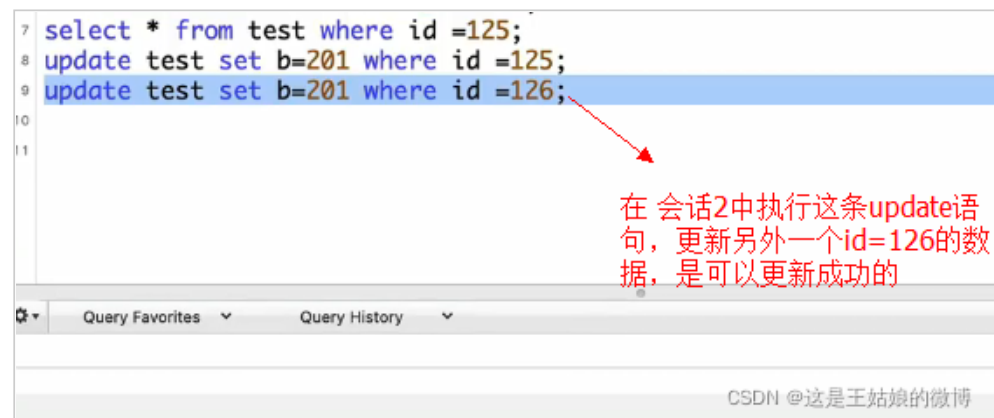
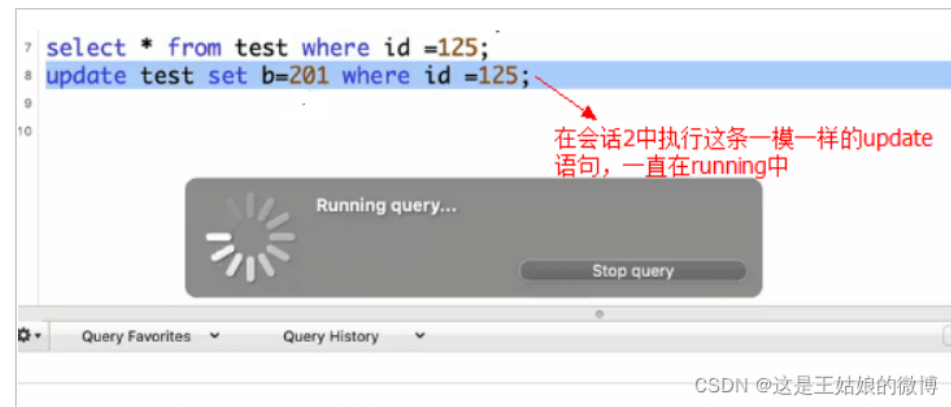
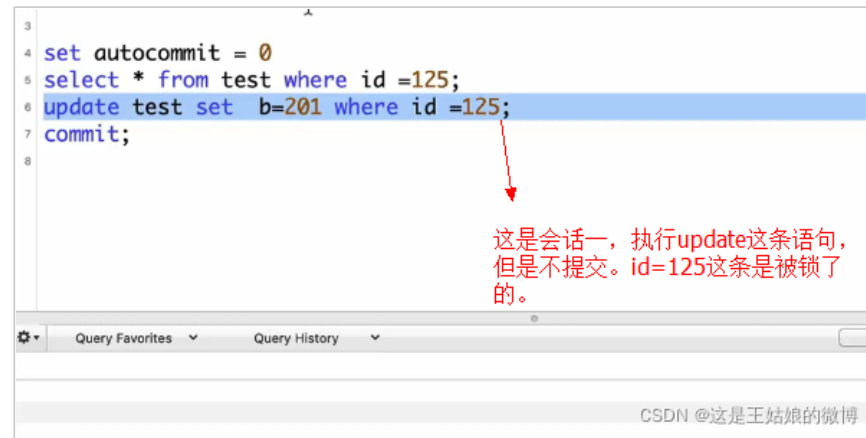
其中会话2的update一直都在Running中，一直到超时结束，或者会话1提交事务后才会Running结束。

可以通过show VARIABLES like "%innodb_lock_wait_timeout%" 查询当前mysql设置的锁超时时间,默认是50秒。

可以通过set innodb_lock_wait_timeout = 60; 设置锁的超时时间。

当第一个会话commit之后，第二个会话的update语句才会执行成功。这代表了innodb用了锁。

那怎么确定是用了行锁呢？



总结:会话一更新id=125的时候, 给这条数据add lock了, 那么在会话2中再次更新id=125的时候, 这条数据是locked中的。这个lock加的是id=125这条记录。此时除了id=125这条之外的, 都是可以成功的, 证明这条默认加的是行锁。

三、InnoDB死锁概念和死锁案例

定义: 当两个或以上的事务相互持有和请求锁, 并形成一个循环的依赖关系, 就会产生死锁。多个事务同时锁定同一个资源时, 也会产生死锁。在一个事务系统中, 死锁是确切存在并且是不能完全避免的。

解决: InnoDB会自动检测事务死锁, 立即回滚其中某个事务, 并且返回一个错误。它根据某种机制来选择那个最简单(代价最小)的事务来进行回滚

死锁场景一之select for update:

产生场景: 两个transaction都有两个select for update, transaction a先锁记录1, 再锁记录2; 而transaction b先锁记录2, 再锁记录1

写锁: for update, 读锁: for my share mode show engine innodb status

验证下死锁的场景:

第一步

第二步

第三步

第四步, 执行完, 下面提示死锁了

| id | a | b |
|-----|-----|-----|
| 101 | 101 | 101 |

[SQL] select * from wnn_test where a=199 for update;
[Err] 1213 - Deadlock found when trying to get lock; try restarting transaction

CSDN @这是王姑娘的微博

第一步更新会话一:

```
1 | start TRANSACTION;  
2 | select * from wnn_test where a=199 for update;
```

第二步更新会话二:

```
1 | start TRANSACTION;  
2 | select * from wnn_test where a=101 for update;
```

第三步更新会话一:

```
1 | select * from wnn_test where a=101 for update;
```

第四步更新会话二;

```
1 | select * from wnn_test where a=199 for update;
```

在更新到第三步和第四步的时候, 已经发生了死锁。

来看下执行的日志:

show engine innodb status;最后一个锁的时间, 锁的表, 引起锁的语句。其中session1被锁 14秒(ACTIVE 14), session 2被锁了10秒(Active 10)

LATEST DETECTED DEADLOCK

2022-03-10 21:43:35 140180758730496

*** (1) TRANSACTION:

TRANSACTION 3448, ACTIVE 14 sec starting index read

mysql tables in use 1, locked 1

LOCK WAIT 5 lock struct(s), heap size 1128, 4 row lock(s)

MySQL thread id 6813, OS thread handle 140179791562496, query id 3196524 112.32.10.127 root executing

select * from wnn_test where a=101 for update

*** (1) HOLDS THE LOCK(S):

RECORD LOCKS space id 41 page no 5 n bits 272 index a of table `test`.`wnn_test` trx id 3448 lock_mode X

Record lock, heap no 200 PHYSICAL RECORD: n_fields 2; compact format; info bits 0

0: len 4; hex 800000c7; asc ;;

1: len 4; hex 800000c7; asc ;;

*** (1) WAITING FOR THIS LOCK TO BE GRANTED:

RECORD LOCKS space id 41 page no 5 n bits 272 index a of table `test`.`wnn_test` trx id 3448 lock_mode X waiting

Record lock, heap no 102 PHYSICAL RECORD: n_fields 2; compact format; info bits 0

0: len 4; hex 80000065; asc e;;

1: len 4; hex 80000065; asc e;;

*** (2) TRANSACTION:

TRANSACTION 3449, ACTIVE 10 sec starting index read

mysql tables in use 1, locked 1

LOCK WAIT 5 lock struct(s), heap size 1128, 4 row lock(s)

MySQL thread id 6815, OS thread handle 140179812636416, query id 3196528 112.32.10.127 root executing

select * from wnn_test where a=199 for update

*** (2) HOLDS THE LOCK(S):

RECORD LOCKS space id 41 page no 5 n bits 272 index a of table `test`.`wnn_test` trx id 3449 lock_mode X

Record lock, heap no 102 PHYSICAL RECORD: n_fields 2; compact format; info bits 0

0: len 4; hex 80000065; asc e;;

1: len 4; hex 80000065; asc e;;

CSDN @这是王姑娘的微博

死锁场景二之两个update

产生场景：两个transaction都有两个update，transaction a先更新记录1，再更新记录2；而transaction b先更新记录2，再更新记录1

| | |
|---|--|
| <pre> 8 9 start TRANSACTION; 10 update wnn_test set b=100 where a=101; 11 12 update wnn_test set b=100 where a=199; 13 14 commit; 15 16 </pre> <p>信息 概况 状态</p> <p>[SQL] update wnn_test set b=100 where a=199; 受影响的行: 0 时间: 3.197ms</p> | <pre> 6 7 8 start TRANSACTION; 9 update wnn_test set b=100 where a=199; 10 11 update wnn_test set b=100 where a=101; 12 13 commit; 14 </pre> <p>信息 概况 状态</p> <p>[SQL] update wnn_test set b=100 where a=101; [Err] 1213 - Deadlock found when trying to get lock; try restarting transaction</p> |
|---|--|

CSDN @这是王姑娘的微博

产生日志:

```

-----
LATEST DETECTED DEADLOCK
-----
2022-03-10 22:08:00 140180758730496
*** (1) TRANSACTION:
TRANSACTION 3458, ACTIVE 13 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 5 lock struct(s), heap size 1128, 4 row lock(s)
MySQL thread id 6825, OS thread handle 140180151219968, query id 3197827 112.32.10.127 root updating
update wnn_test set b=100 where a=199

*** (1) HOLDS THE LOCK(S):
RECORD LOCKS space id 41 page no 5 n bits 272 index a of table `test`.`wnn_test` trx id 3458 lock_mode X
Record lock, heap no 102 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
 0: len 4; hex 80000065; asc e;;
 1: len 4; hex 80000065; asc e;;

*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 41 page no 5 n bits 272 index a of table `test`.`wnn_test` trx id 3458 lock_mode X waiting
Record lock, heap no 200 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
 0: len 4; hex 800000c7; asc ;;
 1: len 4; hex 800000c7; asc ;;

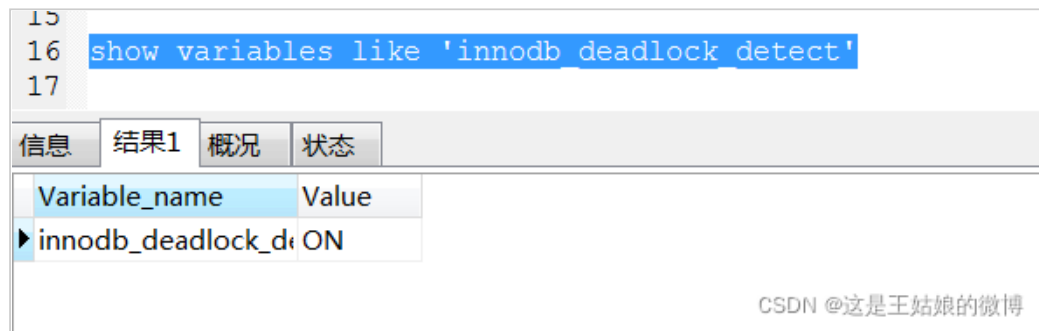
*** (2) TRANSACTION:
TRANSACTION 3459, ACTIVE 9 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 5 lock struct(s), heap size 1128, 4 row lock(s)
MySQL thread id 6826, OS thread handle 140179811579648, query id 3197838 112.32.10.127 root updating
update wnn_test set b=100 where a=101

*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 41 page no 5 n bits 272 index a of table `test`.`wnn_test` trx id 3459 lock_mode X
Record lock, heap no 200 PHYSICAL RECORD: n_fields 2; compact format; info bits 0
 0: len 4; hex 800000c7; asc ;;
 1: len 4; hex 800000c7; asc ;;

```

CSDN @这是王姑娘的微博

注意：仔细查看上面2个例子可以发现一个现象，当2条资源锁住后，再执行第三个会执行成功，但是第四个会提示死锁。在mysql5.7中，执行第三个的时候就会一直在Running状态了，博文使用的是mysql8.0，其中有这个参数 `innodb_deadlock_detect` 可以用于控制 InnoDB 是否执行死锁检测，**当启用了死锁检测时（默认设置），InnoDB 自动执行事务的死锁检测，并且回滚一个或多个事务以解决死锁。InnoDB 尝试回滚更小的事务，事务的大小由它所插入、更新或者删除的数据行数决定。**



那么这个 `innodb_deadlock_detect` 参数，到底要不要启用呢？

对于**高并发的系统**，当大量线程等待同一个锁时，**死锁检测可能会导致性能的下降**。此时，如果禁用死锁检测，而改为依靠参数 `innodb_lock_wait_timeout` 执行发生死锁时的事务回滚可能会更加高效。

通常来说，应该启用死锁检测，并且在应用程序中尽量避免产生死锁，同时对死锁进行相应的处理，例如重新开始事务。

只有在确认死锁检测影响了系统的性能，并且禁用死锁检测不会带来负面影响时，可以尝试关闭 `innodb_deadlock_detect` 选项。另外，**如果禁用了 InnoDB 死锁检测，需要调整参数 `innodb_lock_wait_timeout` 的值**，以满足实际的需求。

四、程序开发过程中应该如何注意避免死锁

锁的本质是资源相互竞争，相互等待，往往是两个(或以上)的Session加锁的顺序不一致

如何有效避免：

在程序中，操作多张表时，尽量以相同的顺序来访问（避免形成等待环路）

批量操作单张表数据的时候，先对数据进行排序（避免形成等待环路） A线程 id: 1,10,20按顺序加锁 B线程id:20,10,1 这种的话就容易锁。

如果可以，大事务化成小事务，甚至不开启事务 `select for update==>insert==>update = insert into update on duplicate key`

尽量使用索引访问数据，避免没有 `where` 条件的操作，避免锁表 有走索引是记录行锁，没走索引是表锁

使用等值查询而不是范围查询查询数据，命中记录，避免间隙锁对并发的影响 1, 10, 20 等值 `where id in (1,10,20)` 范围查询 `id>1 and id<20`

避免在同一时间点运行多个对同一表进行读写的脚本，特别注意加锁且操作数据量比较大的语句；我们经常会有一些定时脚本，避免它们在同一时间点运行

到此这篇关于Mysql锁机制之行锁、表锁、死锁的实现的文章就介绍到这了,更多相关Mysql 行锁、表锁、死锁内容请搜索脚本之家以前的文章或继续浏览下面的相关文章希望大家以后多多支持脚本之家！

原文链接：<https://blog.csdn.net/wnn654321/article/details/123364797>

[关于我们](#) - [广告合作](#) - [联系我们](#) - [免责声明](#) - [网站地图](#) - [投诉建议](#) - [在线投稿](#)

©Copyright 2006-2021 JB51.Net Inc All Rights Reserved. 脚本之家 版权所有