# Mysql锁详解

# 悲观锁与乐观锁

悲观锁与乐观锁是两种常见的资源并发锁设计思路，也是并发编程中一个非常基础的概念。并不是数据库独有，这里只是以数据中实现乐观锁和悲观锁作为例子讲解。

**一、悲观锁**

1、排它锁，当事务在操作数据时把这部分数据进行锁定，直到操作完毕后再解锁，其他事务操作才可操作该部分数据。这将防止其他进程读取或修改表中的数据。

2、实现：大多数情况下依靠数据库的锁机制实现

一般使用 select ...for update 对所选择的数据进行加锁处理，例如select \* from account where name=”Max” for update， 这条sql 语句锁定了account 表中所有符合检索条件（name=”Max”）的记录。本次事务提交之前（事务提交时会释放事务过程中的锁），外界无法修改这些记录。

**for update的注意点**

1.for update 仅适用于InnoDB，并且必须开启事务，在begin与commit之间才生效。

2.要测试for update的锁表情况，可以利用MySQL的Command Mode，开启二个视窗来做测试。

当开启一个事务进行for update的时候，另一个事务也有for update的时候会一直等着，直到第一个事务结束吗？

答：会的。除非第一个事务commit或者rollback或者断开连接，第二个事务会立马拿到锁进行后面操作。

**二、乐观锁**

1、如果有人在你之前更新了，你的更新应当是被拒绝的，可以让用户重新操作。

2、实现：大多数基于数据版本（Version）记录机制实现

具体可通过给表加一个版本号或时间戳字段实现，当读取数据时，将version字段的值一同读出，数据每更新一次，对此version值加一。当我们提交更新的时候，判断当前版本信息与第一次取出来的版本值大小，如果数据库表当前版本号与第一次取出来的version值相等，则予以更新，否则认为是过期数据，拒绝更新，让用户重新操作。

例子：

|  |
| --- |
| 1. SELECT data AS old\_data, version AS old\_version FROM …where id=1;  2. 根据获取的数据进行业务操作，得到new\_data和new\_version  new\_version= old\_version+1  3. UPDATE SET data = new\_data, version = new\_version WHERE version = old\_version and id=1  if (updated row > 0) {  // 乐观锁获取成功，操作完成  } else {  // 乐观锁获取失败，回滚并重试  } |

**乐观锁与悲观锁的选择?**

两种锁各有优缺点，不可认为一种好于另一种，像**乐观锁适用于写比较少的情况下**，即冲突真的很少发生的时候，这样可以省去了锁的开销，加大了系统的整个吞吐量。但**如果经常产生冲突**，上层应用会不断的进行retry，这样反倒是降低了性能，**所以这种情况下用悲观锁就比较合适。**

# 行级锁、表级锁、页面锁

根据锁的粒度或范围划分：行级锁、表级锁、页面锁

**表级锁**: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**:开锁大，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。

**页面锁**: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。

**各引擎对锁的支持情况：**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 行锁 | 表锁 | 页锁 |
| MYISAM |  | √ |  |
| BDB |  | √ | √ |
| InnoDB | √ | √ |  |

**优缺点：**

**表级锁:**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**:开锁大，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。

**页面锁:**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。

行级锁都是基于索引的，如果一条SQL语句用不到索引是不会使用行级锁的，会使用表级锁。行级锁的缺点是：由于需要请求大量的锁资源，所以速度慢，内存消耗大。

# 死锁

死锁是指两个或两个以上的进程在执行过程中,因争夺资源而造成的一种互相等待的现象,若无外力作用,它们都将无法推进下去.此时称系统处于死锁状态或系统产生了死锁,这些永远在互相等的进程称为死锁进程.

**死锁产生的四个必要条件**

•互斥条件：指进程对所分配到的资源进行排它性使用，即在一段时间内某资源只由一个进程占用。如果此时还有其它进程请求资源，则请求者只能等待，直至占有资源的进程用毕释放

•请求和保持条件：指进程已经保持至少一个资源，但又提出了新的资源请求，而该资源已被其它进程占有，此时请求进程阻塞，但又对自己已获得的其它资源保持不放

•不剥夺条件：指进程已获得的资源，在未使用完之前，不能被剥夺，只能在使用完时由自己释放

•环路等待条件：指在发生死锁时，必然存在一个进程——资源的环形链，即进程集合{P0，P1，P2，···，Pn}中的P0正在等待一个P1占用的资源；P1正在等待P2占用的资源，……，Pn正在等待已被P0占用的资源

这四个条件是死锁的必要条件，只要系统发生死锁，这些条件必然成立，而只要上述条件之一不满足，就不会发生死锁。

如果能由结论推出 条件，但由条件推不出结论。此条件为必要条件

**如何尽可能避免死锁:**

1）以固定的顺序访问表和行。比如两个更新数据的事务，事务A 更新数据的顺序 为1，2；事务B更新数据的顺序为2，1。这样更可能会造成死锁。

2）大事务拆小。大事务更倾向于死锁，如果业务允许，将大事务拆小。

3）在同一个事务中，尽可能做到一次锁定所需要的所有资源，减少死锁概率。

4）降低隔离级别。如果业务允许，将隔离级别调低也是较好的选择，比如将隔离级别从RR调整为RC，可以避免掉很多因为gap锁造成的死锁。

5）为表添加合理的索引。可以看到如果不走索引将会为表的每一行记录添加上锁，死锁的概率大大增大。

**如何查看MySQL数据库的死锁信息**

打印mysql最近一次的死锁信息

SHOW ENGINE INNODB STATUS；

**死锁案例**

参考博文：<https://blog.csdn.net/varyall/article/details/80209488>

**解除死锁**

1. 查看在锁的事务，

SELECT \* FROM INFORMATION\_SCHEMA.INNODB\_TRX;

INNODB\_TRX为在锁的事务表

1. 杀死进程id（就是上面命令的trx\_mysql\_thread\_id列）

kill id

**其它查看事务命令**

1：查看当前的事务

SELECT \* FROM INFORMATION\_SCHEMA.INNODB\_TRX;

2：查看当前锁定的事务

SELECT \* FROM INFORMATION\_SCHEMA.INNODB\_LOCKS;

3：查看当前等锁的事务

SELECT \* FROM INFORMATION\_SCHEMA.INNODB\_LOCK\_WAITS;