## 锁分类

### 性能区分锁

#### 乐观锁

通过版本对比或者CAS机制实现，适用于读操作比较多的场景。

#### **悲观锁**

适用于写操作比较多的场景，频繁加锁影响性能；

### 操作分锁

#### 读锁

读锁也叫共享锁，读的时候其他事务不能做修改操作，其他可以查询。

select \* from T where id=1 **lock in share model;**

#### 写锁

写锁也叫排它锁，写锁会阻塞其他读写操作。

select \* from T where id=1 **for update;**

#### 意向锁(mysql独有)

意向锁（Intention Lock）：又称I锁，针对表锁，主要是为了提高加表锁的效率，是mysql数据库自己加的。当有事务给表的数据行加了共享锁或排他锁，同时会给表设置一个标识，代表已经有行锁了，其他事务要想对表加表锁时，就不必逐行判断有没有行锁可能跟表锁冲突了，直接读这个标识就可以确定自己该不该加表锁。特别是表中的记录很多时，逐行判断加表锁的方式效率很低。而这个标识就是意向锁。

意向锁主要分为：

意向共享锁，IS锁，对整个表加共享锁之前，需要先获取到意向共享锁。

意向排他锁，IX锁，对整个表加排他锁之前，需要先获取到意向排他锁。

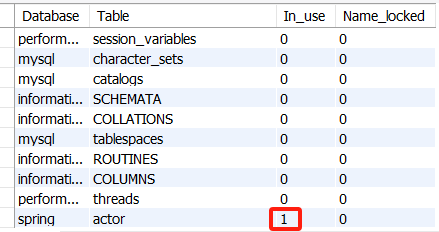
### 数据操作区分锁

#### 表锁

开销小，加锁快，但是并发差，适合整表迁移场景。

lock table actor write(read); -给actor加写(读锁)锁

show open tables; --查看表锁情况



unlock tables; --解锁

#### 页锁

只有**BDB存储引擎**支持页锁，页锁就是在页的粒度上进行锁定，锁定的数据资源比行锁要多，因为一个页中可以有多个行记录。当我们使用页锁的时候，会出现数据浪费的现象，但这样的浪费最多也就是一个页上的数据行。

页锁的开销介于表锁和行锁之间，会出现死锁。锁定粒度介于表锁和行锁之间，并发度一般。

#### 行锁

开销大，加锁慢，会死锁，并发性能高。

InnoDB相对于MYISAM的最大不同有两点：

* InnoDB支持事务（TRANSACTION）
* InnoDB支持行级锁

InnoDB的行锁实际是针对索引的，如果不是针对索引加锁，会升级表锁。

备注：**RR会导致表锁，RC不会**

例如以下操作会导致表锁：

select \* from T where name=’ak’ **for update;**

**PS：关于RR级别行锁升级为表锁的原因分析**

因为在RR隔离级别下，需要解决不可重复读和幻读问题，所以在遍历扫描聚集索引记录时，为了防止扫描过的索引被其它事务修改(不可重复读问题)或间隙被其它事务插入记录(幻读问题)，从而导致数据不一致，所以MySQL的解决方案就是把所有扫描过的索引记录和间隙都锁上。

这里要注意，并不是直接将整张表加表锁，因为不一定能加上表锁，可能会有其它事务锁住了表里的其它行记录。

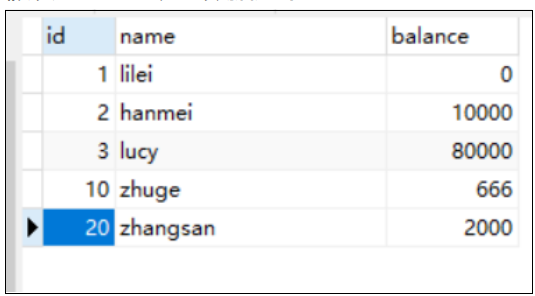
### 其他锁

#### 间隙锁(Gap Lock)

间隙锁，锁的就是两个值之间的空隙，间隙锁是在可重复读隔离级别下才会生效。

Mysql默认级别是repeatable-read，有幻读问题，间隙锁是可以解决幻读问题的。

假设account表里数据如下：



那么间隙就有 id 为 (3,10)，(10,20)，(20,正无穷) 这三个区间，在Session\_1下面执行如下sql：

**select \* from account where id = 18 for update;**

则其他Session没法在这个(10,20)这个间隙范围里插入任何数据。

如果执行下面这条sql：

**select \* from account where id = 25 for update;**

则其他Session没法在这个(20,正无穷)这个间隙范围里插入任何数据。

也就是说，**只要在间隙范围内锁了一条不存在的记录会锁住整个间隙范围，不锁边界记录，这样就能防止其它Session在这个间隙范围内插入数据，就解决了可重复读隔离级别的幻读问题**。

#### 临键锁(Next-key Locks)

Next-Key Locks是行锁与间隙锁的组合。

## 锁总结

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update、insert、delete操作会自动给涉及的表加写锁。

InnoDB在执行查询语句SELECT时(非串行隔离级别)，不会加锁。但是update、insert、delete操作会加行锁。

另外，读锁会阻塞写，但是不会阻塞读。而写锁则会把读和写都阻塞。

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下，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。

## 锁等待分析

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。

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长的一次所花时间

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）

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制定优化计划。

## 锁相关数据表

‐‐ 查看事务

select \* from INFORMATION\_SCHEMA.INNODB\_TRX;

‐‐ 查看锁，8.0之后需要换成这张表performance\_schema.data\_locks

select \* from INFORMATION\_SCHEMA.INNODB\_LOCKS;

‐‐ 查看锁等待，8.0之后需要换成这张表performance\_schema.data\_lock\_waits

select \* from INFORMATION\_SCHEMA.INNODB\_LOCK\_WAITS;

‐‐ 释放锁，trx\_mysql\_thread\_id可以从INNODB\_TRX表里查看到

kill trx\_mysql\_thread\_id

‐‐ 查看锁等待详细信息

show engine innodb status;

## 死锁问题分析

set tx\_isolation='repeatable‐read';

Session\_1执行：select \* from account where id=1 for update;

Session\_2执行：select \* from account where id=2 for update;

Session\_1执行：select \* from account where id=2 for update;

Session\_2执行：select \* from account where id=1 for update;

查看近期死锁日志信息：show engine innodb status;

大多数情况mysql可以自动检测死锁并回滚产生死锁的那个事务，但是有些情况mysql没法自动检测死锁，这种情况我们可以通过日志分析找到对应事务线程id，可以通过kill杀掉。

## 锁优化实践

* 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
* 合理设计索引，尽量缩小锁的范围
* 尽可能减少检索条件范围，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，涉及事务加锁的sql尽量放在事务最后执行
* 尽可能用低的事务隔离级别