**https://www.cnblogs.com/leedaily/p/8378779.html**

# 小结

本文重点介绍了MySQL中MyISAM表级锁和InnoDB行级锁的实现特点，并讨论了两种存储引擎经常遇到的锁问题和解决办法。

**对于MyISAM的表锁，主要讨论了以下几点：**

（1）共享读锁（S）之间是兼容的，但共享读锁（S）与排他写锁（X）之间，以及排他写锁（X）之间是互斥的，也就是**说读和写是串行的**。

（2）在一定条件下，MyISAM允许查询和插入并发执行，我们可以利用这一点来解决应用中对同一表查询和插入的锁争用问题。

（3）MyISAM**默认的锁调度机制是写优先**，这并不一定适合所有应用，用户可以通过设置LOW_PRIORITY_UPDATES参数，或在INSERT、UPDATE、DELETE语句中指定LOW_PRIORITY选项来调节读写锁的争用。

（4）由于表锁的锁定粒度大，读写之间又是串行的，因此，**如果更新操作较多，MyISAM表可能会出现严重的锁等待**，可以考虑采用InnoDB表来减少锁冲突。

**对于InnoDB表，本文主要讨论了以下几项内容：**

（1）InnoDB的**行锁是基于索引实现的，如果不通过索引访问数据，InnoDB会使用表锁。**

（2）介绍了InnoDB间隙锁（Next-key)机制，以及InnoDB使用间隙锁的原因。

在不同的隔离级别下，InnoDB的锁机制和一致性读策略不同。

在了解InnoDB锁特性后，用户可以通过设计和SQL调整等措施减少锁冲突和死锁，包括：

- **尽量使用较低的隔离级别； 精心设计索引**，并尽量使用索引访问数据，使加锁更精确，从而减少锁冲突的机会；
- 选择合理的事务大小，**小事务发生锁冲突的几率也更小**；

- 给记录集显式加锁时，最好一次性请求足够级别的锁。比如要修改数据的话，最好直接申请排他锁，而不是先申请共享锁，修改时再请求排他锁，这样容易产生死锁；
- 不同的程序访问一组表时，应尽量约定以相同的顺序访问各表，**对一个表而言，尽可能以固定的顺序存取表中的行**。这样可以大大减少死锁的机会；
- 尽量用相等条件访问数据，这样可以避免间隙锁对并发插入的影响； 不要申请超过实际需要的锁级别；除非必须，查询时不要显示加锁；
- 对于一些特定的事务，可以使用表锁来提高处理速度或减少死锁的可能。

## 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4属性，通常称为事务的ACID属性。

- 原子性（Actomicity）：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。**--使用Redo& Undo**
- 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，以操持完整性；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。 **--Undo**
- **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。----这里就使用数据库的Lock**
- 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。**--Redo**

**而事务的ACID是通过InnoDB日志和锁来保证。事务的隔离性是通过数据库锁的机制实现的，持久性通过redo log（重做日志）来实现，原子性和一致性通过Undo log来实现。UndoLog的原理很简单，为了满足事务的原子性，在操作任何数据之前，首先将数据备份到一个地方（这个存储数据备份的地方称为UndoLog）。然后进行数据的修改。如果出现了错误或者用户执行了ROLLBACK语句，系统可以利用Undo Log中的备份将数据恢复到事务开始之前的状态。**

**和Undo Log相反，RedoLog记录的是新数据的备份。在事务提交前，只要将RedoLog持久化即可，不需要将数据持久化。当系统崩溃时，虽然数据没有持久化，但是RedoLog已经持久化。系统可以根据RedoLog的内容，将所有数据恢复到最新的状态。**

## 什么是锁?

- 锁用于在多个事务访问同一个对象时根据这些操作访问同一对象的现后次序给事务排序
- 就是多人访问同一个东西时候给人排序操作

**不同数据库锁的实现：**

- InnoDB行级锁
- MyISam锁

**Lock与Latch的区别**

- 对象： 事务/线程
- 保护：数据库对象/内存结构对象
- 持续时间：长/短
- 模式：表锁行锁/互斥
- 死锁：有/无

| | 读数据一致性 | 脏读 | 不可重复读 | 幻读 |
|---|---|---|---|---|
| 未提交读 Read uncommitted | 最低级别，只能保证不读取物理上损坏的数据 | √ | √ | √ |
| 已提交读 Read committed | 语句级 | × | √ | √ |
| 可重复读 Repeatable read | 事务级 | × | × | √ |
| 可序列化 Serializable | 最高级别，事务级 | × | × | × |



## InnoDB行级锁

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

**行级锁：**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。

**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

- 锁的模式
  - **排他锁是针对修改操作，又称写锁**
  - **共享锁是针对查询操作，又称读锁，普通查询没有任何锁机制**
  - 表锁
    - 意向排他锁IX--锁表，可以让并发时别人删除马上发现有人在操作，马上报错，提交效率，
    - 意向共享锁IS --- 让别人也可以访问这个记录
  - 行锁

    - 排他锁X
    - 共享锁S

> **InnoDB行锁模式兼容性列表:**
>
> | | X | IX | S | IS |
> |---|---|---|---|---|
> | X | 冲突 | 冲突 | 冲突 | 冲突 |
> | IX | 冲突 | 兼容 | 冲突 | 兼容 |
> | S | 冲突 | 冲突 | 兼容 | 兼容 |
> | IS | 冲突 | 兼容 | 兼容 | 兼容 |
>
> 

- 行锁实现方式
    - 记录锁X Record-lock
        - 发生在修改一条记录时候
    - 间隙锁
        - 修改记录时，锁住表的间隙，防止出现幻读，是针对可重复级别的。
            - 幻读的问题存在是因为新增或者更新操作，这时如果进行范围查询的时候（加锁查询），会出现不一致的问题
- MDL锁
    - 解决或保证DDL(drop)操作与DML(select语句)操作之间一致性
- 死锁现象
    - 两个事务都进行更新数据，第一个事务先X锁住了第二个事务想第二个修改的，第二个事务也锁住了第一个事务修改的第二个，造成循环
        - 解决：会把最短的事务回滚回去
    - 加锁顺序产生循环
    - 减少死锁方法：
        - 自动死锁检测，优先回滚小事务
        - 锁超时设置
        - 尽快提交事务
- 怎么避免mysql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比较好的选择，比如将隔离级别从RR调整为RC,可以避免很多因为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，死锁的概率大大增加。