**数据库锁机制**

**锁：资源的并发访问控制**

**(1)锁的分类：**

**①操作类型分类：读锁/写锁**

**②锁定粒度分类：表锁/行锁**

**读锁(共享S锁)：针对同一份数据，多个读操作可以同时进行而不会相互阻塞。**

**写锁(排他X锁)：当前写操作没有完成前，它会阻塞其他写锁和读锁。**

**表锁：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。**

**行锁：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，并发度最高。**

**(2)SQL命令：**

**①查看表的锁定状态：show open tables;#In use = 0代表该表上没有锁**

**②手动增加表锁(不会自动释放，必须手动释放)：lock table 表名 read/write;**

**③手动释放表锁(不会自动释放，必须手动释放)：unlock tables;**

**注意：只会释放当前事务锁住的表**

**④查看系统上的表锁定情况：**

**show status like ‘table%’;**

**Table\_locks\_immediate：产生表级锁定的次数**

**Table\_locks\_waited：出现表级锁定导致的等待的次数，此值较高说明存在较为严 重的表级锁争抢情况**

**(3)存储引擎：**

**①MyISAM存储引擎：**

**1.MyISAM在执行查询语句前，会自动给涉及到的所有表加读锁(读完自动释放)，其 他线程可以读，写操作阻塞。在执行更新操作前，会自动给涉及到的表加写锁(写 完自动释放)，其他线程的读写操作均阻塞。阻塞将会一直持续到锁释放。**

**2.MyISAM适合偏读，并且对事务要求不高的场景(不支持事务)。锁调度时写优先， 即当表锁释放时，写操作优先获得锁。如果有大量的写操作将导致读操作很难获 得锁，从而造成永远阻塞。**

**②InnoDB存储引擎：**

**与MyISAM存储引擎最大的不同：支持事务、行锁**

**事务：事务是由一组SQL语句组成的逻辑处理单元。**

**1. 事务的四个特性(ACID)：**

**(1) 原子性A：**

**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。**

1. **一致性C：**

**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。**

1. **隔离性I：**

**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。**

1. **持久性D：**

**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。**

**事务的原子性是通过undo loh来实现的**

**事务的持久性是通过redo log来实现的**

**事务的隔离性是通过读写锁+MVCC来实现的**

**事务的一致性是通过原子性、持久性、隔离性来实现的**

**undo log：**

**undo log是为了实现事务的原子性，在MySQL数据库InnoDB存储引擎中， 通过undo log来实现多版本并发控制(简称MVCC)。**

**在操作任何数据之前，首先将数据备份到一个地方，即undo log。然后进行数 据的修改。如果出现了错误或者用户执行了回滚操作，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。**

**注意：undo log是逻辑日志，可以理解为：**

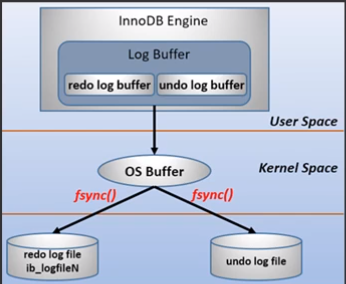
**当delete一条记录时，undo log中会记录一条对应的insert记录**

**当insert一条记录时，undo log中会记录一条对应的delete记录**

**当update一条记录时，undo log中会记录一条相反的update记录**

**redo log：**

**redo log记录的是新数据的备份。在事务提交前，只需要将redo log持久化即 可，不需要将数据持久化(虽然事务是一组SQL语句同时提交，但是SQL仍然 是单条执行的。在写数据时，是先写入到OS Buffer中，然后再fsync 到磁盘 上的)。当系统崩溃时，虽然数据还没有从OS Buffer持久化到磁盘上，但是redo log已经持久化。操作系统可以根据redo log的内容，将所有数据恢复到最新 的状态。**



1. **并发事务处理带来的问题：**
2. **更新丢失：**

**事务A在修改某一行数据时，事务B也在同时修改，导致事务A的更新丢失。**

1. **脏读：**

**事务A读取到了事务B已修改但尚未提交的数据，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。**

1. **不可重复读：**

**事务A读取到了事务B已经提交的修改数据，事务A前后两次读取到的数据 内容不一致。不符合隔离性。**

1. **幻读：**

**事务A读取到了事务B已经提交的新增数据。事务A前后两次读取到的数据 数量不一致。不符合隔离性**

**事务的隔离级别：**

**理论上来说事务之间的执行不应该相互产生影响，其对数据库的影响应该和它 们串行执行时一样。然而完全的隔离性(串行)会导致系统并发性能很低，降低对资 源的利用率，因而实际上对隔离性的要求会有所放宽，这也会一定程度造成对数 据库一致性要求降低。SQL标准为事务定义了不同的隔离级别，从低到高依次是：**

**读未提交、读已提交、可重复读(MySQL默认)、串行化**

**查看数据库当前的事务隔离级别：SHOW VARIABLES LIKE 'tx\_isolation';**

**脏读 不可重复读 幻读**

1. **读未提交： 是 是 是**
2. **读已提交： 否 是 是**
3. **可重复读： 否 否 是**
4. **串行化： 否 否 否**

**隔离性实现原理：锁**

1. **共享锁(S)：**

**共享锁定是将对象数据变成只读形式，其他事务不能进行更新。也称读取锁定。**

1. **排它锁(X)：**

**排他锁定是当执行更新操作的时候，其他事务不能读取或更新该数据。称写入 锁定。排它锁不能和其他锁共存。【执行更新操作时默认会加排它锁(如果不走索引升级为表锁)，解决了丢失更新问题】**

**注意：可以同时存在事务A的S锁和X锁，但是如果已经存在了其他事务的S锁， 则事务A不能再加X锁，除非所有其他事务释放S锁。**

**例如：同时开三个窗口，分别开启事务A、事务B、事务C。**

**事务A对id=1的记录加S锁(lock in share mode，下同)->Sa**

**事务B对id=1的记录加S锁->Sb**

**事务C对id=1的记录加S锁->Sc**

**此时id=1的记录存在Sa、Sb、Sc**

**事务A对id=1的记录加X锁，此时由于存在Sb、Sc，所以Xa阻塞**

**事务B提交，此时由于存在Sc，所以Xa阻塞**

**事务C提交，此时由于只有Sa，所以Xa成功**

**注意：锁定粒度：**

**锁定对象的大小是锁的粒度：记录、表、数据库**

**锁定粒度越大，效率越低，锁定粒度越小，效率越高**