**一、为什么要加锁**

锁机制用于管理对共享资源的并发访问。

当多个用户并发地存取数据时，在数据库中就可能会产生多个事务同时操作同一行数据的情况，若对并发操作不加控制就可能会读取和存储不正确的数据，破坏数据的一致性。

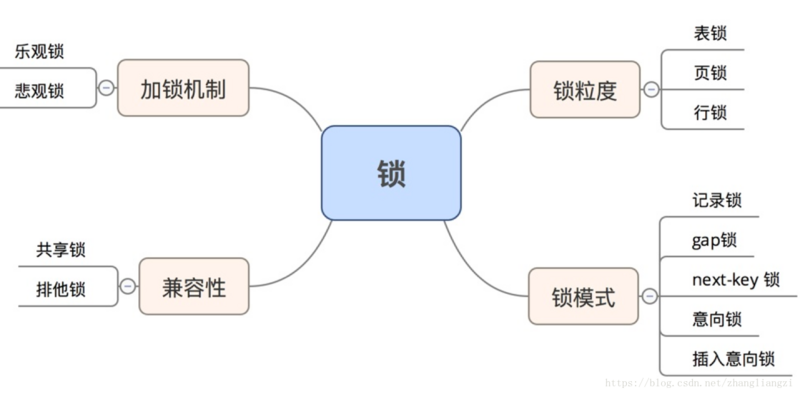
一种典型的并发问题——丢失更新（其他锁问题及解决方法会在后面说到）：

注：RR默认隔离级别下，为更清晰体现时间先后，暂时忽略锁等待，不影响最终效果~

| **时间点** | **事务A** | **事务B** |
| --- | --- | --- |
| 1 | 开启事务A |  |
| 2 |  | 开启事务B |
| 3 | 查询当前商品S库存为100 |  |
| 4 |  | 查询当前商品S库存为100 |
| 5 | 业务逻辑处理，确定要将商品S库存增加10，故更新库存为110（update stock set amount=110 where sku\_id=S;） |  |
| 6 |  | 业务逻辑处理，确定要将商品S库存增加20，故更新库存为120（update stock set amount=120 where sku\_id=S;） |
| 7 | 提交事务A |  |
| 8 |  | 提交事务B |

**异常结果：**商品S库存更新为120，但实际上针对商品S进行了两次入库操作，最终商品S库存应为100+10+20=130，但实际结果为120，首先提交的事务A的更新『丢失了』！！！所以就需要锁机制来保证这种情况不会发生。

**二、InnoDB锁类型概述**



简介（后面会分别详细说到）：

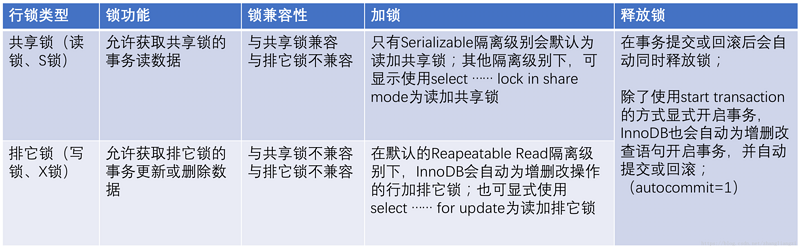
**1、乐观锁与悲观锁是两种并发控制的思想，可用于解决丢失更新问题：**

乐观锁会“乐观地”假定大概率不会发生并发更新冲突，访问、处理数据过程中不加锁，只在更新数据时再根据版本号或时间戳判断是否有冲突，有则处理，无则提交事务；

悲观锁会“悲观地”假定大概率会发生并发更新冲突，访问、处理数据前就加排他锁，在整个数据处理过程中锁定数据，事务提交或回滚后才释放锁；

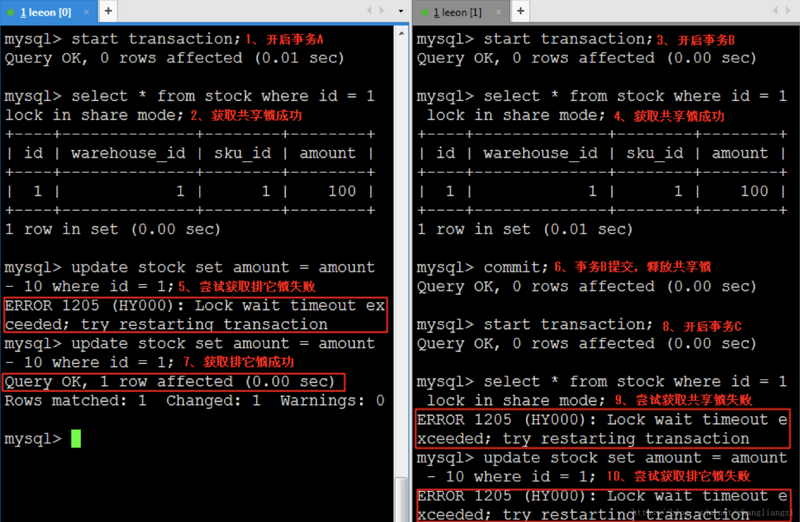
**2、InnoDB支持多种锁粒度，默认使用行锁，锁粒度最小，锁冲突发生的概率最低，支持的并发度也最高，但系统消耗成本也相对较高；**  
**3、共享锁与排他锁是InnoDB实现的两种标准的行锁；**  
**4、InnoDB有三种锁算法——记录锁、gap间隙锁、还有结合了记录锁与间隙锁的next-key锁，InnoDB对于行的查询加锁是使用的是next-key locking这种算法，一定程度上解决了幻读问题；**  
**5、意向锁是为了支持多种粒度锁同时存在；（1.0版本不重点介绍，如有兴趣可参看知乎推荐回答[https://www.zhihu.com/questio...](https://www.zhihu.com/question/51513268" \t "_blank)）**

**三、行锁详解**

InnoDB默认使用行锁，实现了两种标准的行锁——共享锁与排他锁；  


**注意：**  
1、除了显式加锁的情况，其他情况下的加锁与解锁都无需人工干预。  
2、InnoDB所有的行锁算法都是基于索引实现的，锁定的也都是索引或索引区间（这一点会在第六章节『锁算法』中详细说到）；

**共享锁与排它锁兼容性示例（使用默认的RR隔离级别，图中数字从小到大标识操作执行先后顺序）：**



**四、当前读与快照读**

**1、当前读：**即加锁读，读取记录的最新版本，会加锁保证其他并发事务不能修改当前记录，直至获取锁的事务释放锁；

使用当前读的操作主要包括：显式加锁的读操作与插入/更新/删除等写操作，如下所示：

**select** \* **from** **table** **where** ? **lock** **in** **share** **mode**;

**select** \* **from** **table** **where** ? **for** **update**;

**insert** **into** **table** **values** (…);

**update** **table** **set** ? **where** ?;

**delete** **from** **table** **where** ?;

注：当Update SQL被发给MySQL后，MySQL Server会根据where条件，读取第一条满足条件的记录，然后InnoDB引擎会将第一条记录返回，并加锁，待MySQL Server收到这条加锁的记录之后，会再发起一个Update请求，更新这条记录。一条记录操作完成，再读取下一条记录，直至没有满足条件的记录为止。因此，Update操作内部，就包含了当前读。同理，Delete操作也一样。Insert操作会稍微有些不同，简单来说，就是Insert操作可能会触发Unique Key的冲突检查，也会进行一个当前读。

**2、快照读**：即不加锁读，读取记录的快照版本而非最新版本，通过MVCC实现；

InnoDB默认的RR事务隔离级别下，不显式加『lock in share mode』与『for update』的『select』操作都属于快照读，保证事务执行过程中只有第一次读之前提交的修改和自己的修改可见，其他的均不可见；

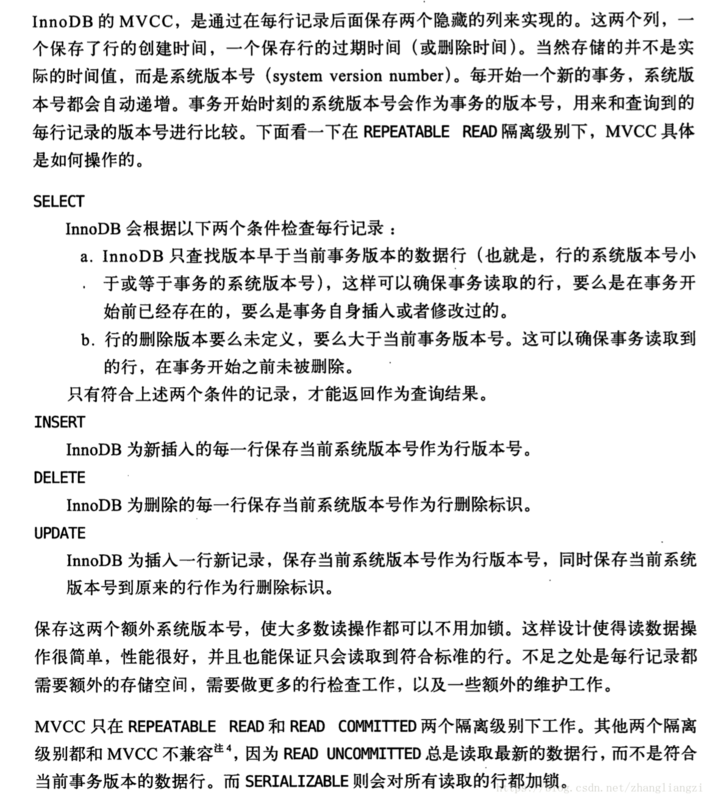
**五、MVCC**

MVCC『多版本并发控制』，与之对应的是『基于锁的并发控制』；

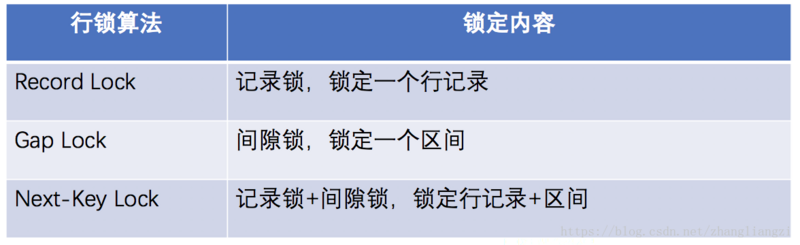
MVCC的最大好处：读不加任何锁，读写不冲突，对于读操作多于写操作的应用，极大的增加了系统的并发性能；

InnoDB默认的RR事务隔离级别下，不显式加『lock in share mode』与『for update』的『select』操作都属于快照读，使用MVCC，保证事务执行过程中只有第一次读之前提交的修改和自己的修改可见，其他的均不可见；

关于InnoDB MVCC的实现原理，在《高性能Mysql》一书中有一些说明，网络上也大多沿用这一套理论，但这套理论与InnoDB的实际实现还是有一定差距的，但不妨我们通过它初步理解MVCC的实现机制，所以我在此贴上此书中的说明；

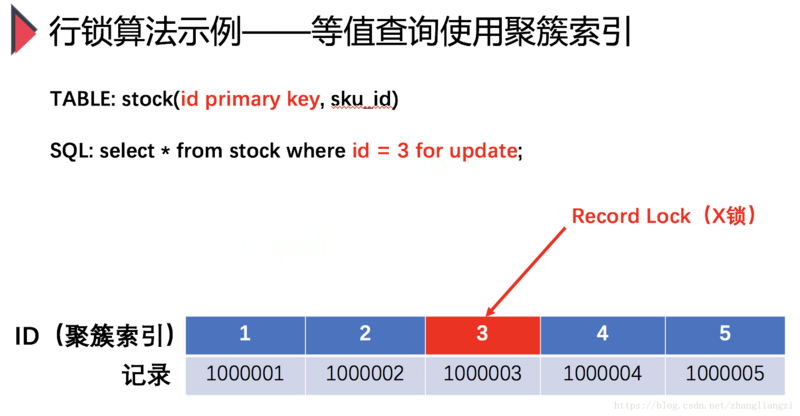


**六、锁算法**

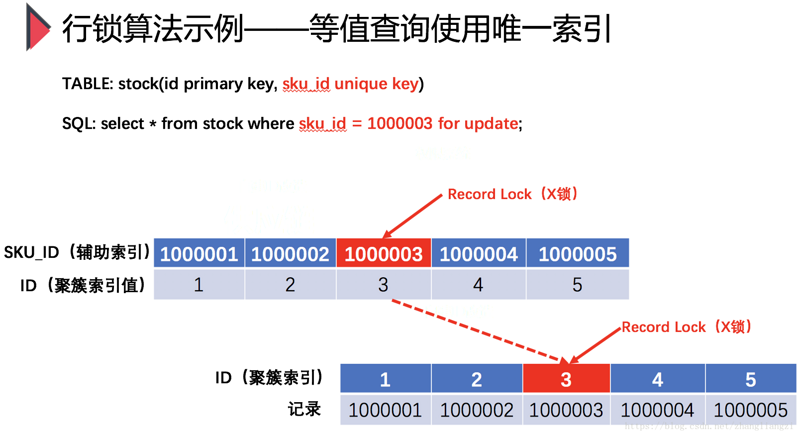
InnoDB主要实现了三种行锁算法：  


InnoDB所有的行锁算法都是基于索引实现的，锁定的也都是索引或索引区间；

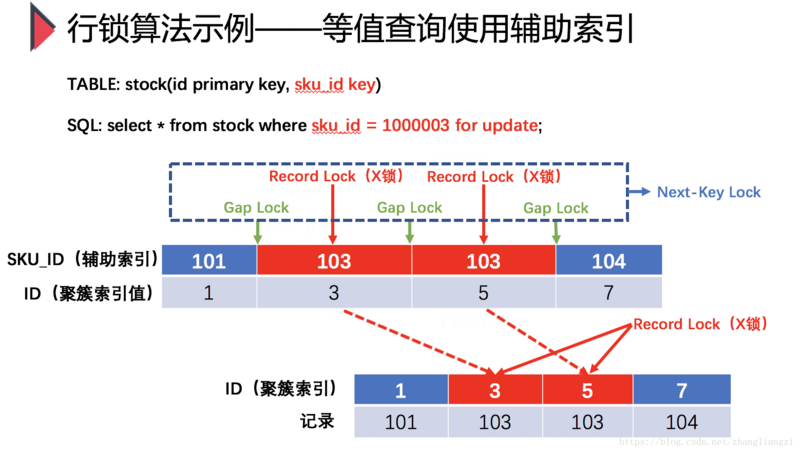
不同的事务隔离级别、不同的索引类型、是否为等值查询，使用的行锁算法也会有所不同；下面仅以InnoDB默认的RR隔离级别、等值查询为例，介绍几种行锁算法：  


**1、等值查询使用聚簇索引**  


注： InnoDB表是索引组织表，根据主键索引构造一棵B+树，叶子节点存放的是整张表的行记录数据，且按主键顺序存放；我这里做了一个表格模拟主键索引的叶子节点，使用主键索引查询，就会锁住相关主键索引，锁住了索引也就锁住了行记录，其他并发事务就无法修改此行数据，直至提交事务释放锁，保证了并发情况下数据的一致性；

**2、等值查询使用唯一索引**  


注：辅助索引的叶子节点除了存放辅助索引值，也存放了对应主键索引值；锁定时会锁定辅助索引与主键索引；

**3、等值查询使用辅助索引**  


注：Gap锁，锁定的是索引记录之间的间隙，是防止幻读的关键；如果没有上图中绿色标识的Gap Lock，其他并发事务在间隙中插入了一条记录如：『insert into stock (id,sku\_id) values(2,103);』并提交，那么在此事务中重复执行上图中SQL，就会查询出并发事务新插入的记录，即出现幻读；（幻读是指在同一事务下，连续执行两次同样的SQL语句可能导致不同的结果，第二次的SQL语句可能返回之前不存在的行记录）加上Gap Lock后，并发事务插入新数据前会先检测间隙中是否已被加锁，防止幻读的出现；

更多锁示例可参看博客：[https://yq.aliyun.com/article...](https://yq.aliyun.com/articles/108095?t=t1" \t "_blank)  
更多锁算法详解可参看何博士博客：[http://hedengcheng.com/?p=771](http://hedengcheng.com/?p=771" \t "_blank)

**七、锁问题**

MySQL锁会带来如下几种问题，如果能解决他们，就可以保证并发情况下不会出现问题；

| **锁问题** | **锁问题描述** | **会出现锁问题的隔离级别** | **解决办法** |
| --- | --- | --- | --- |
| 脏读 | 一个事务中会读到其他并发事务未提交的数据，违反了事务的隔离性； | Read Uncommitted | 提高事务隔离级别至Read Committed及以上； |
| 不可重复读 | 一个事务会读到其他并发事务已提交的数据，违反了数据库的一致性要求；可能出现的问题为幻读，幻读是指在同一事务下，连续执行两次同样的SQL语句可能导致不同的结果，第二次的SQL语句可能返回之前不存在的行记录； | Read Uncommitted、Read Committed | 默认的RR隔离级别下 ，解决办法分为两种情况：1、当前读：Next-Key Lock机制对相关索引记录及索引间隙加锁，防止并发事务修改数据或插入新数据到间隙；（详情参见第六章节『锁算法』）2、版本读：MVCC，保证事务执行过程中只有第一次读之前提交的修改和自己的修改可见，其他的均不可见；提高事务隔离级别至Serializable； |
| 丢失更新 | 见章节一中描述； | Read Uncommitted、Read Committed、Repeatable Read | 默认的RR隔离级别下 ，解决办法分为两种情况：1、乐观锁：数据表增加version字段，读取数据时记录原始version，更新数据时，比对version是否为原始version，如不等，则证明有并发事务已更新过此行数据，则可回滚事务后重试直至无并发竞争；2、悲观锁：读加排他锁，保证整个事务执行过程中，其他并发事务无法读取相关记录，直至当前事务提交或回滚释放锁； |

注：其实InnoDB默认的RR事务隔离级别已经为我们做了大多数的事，业务中更多需要关心『丢失更新』这种问题，通常使用乐观锁方式解决；我们在读操作时一般不会使用加锁读，但MVCC并不能完全解读幻读问题，其他并发事务是可以插入符合当前事务查询条件的数据，只是当前事务因为读快照数据无法查看到，这种情况下应该使用唯一索引等方式保证不会重复插入重复的业务数据，在此不再赘述~

**参考及推荐**

书籍：《高性能MySQL》  
书籍：《MySQL技术内幕:InnoDB存储引擎》（强烈推荐）  
锁算法示例：[https://yq.aliyun.com/article...](https://yq.aliyun.com/articles/108095?t=t1)  
MySQL 加锁处理分析：[http://hedengcheng.com/?p=771](http://hedengcheng.com/?p=771" \t "_blank)  
知乎MySQL锁总结：[https://zhuanlan.zhihu.com/p/...](https://zhuanlan.zhihu.com/p/29150809" \t "_blank)  
知乎InnoDB意向锁讨论：[https://www.zhihu.com/questio...](https://www.zhihu.com/question/51513268)  
MySQL使用锁解决并发下的更新丢失问题：[https://segmentfault.com/a/11...](https://segmentfault.com/a/1190000012939310)