1. 处理并发事务的两种方式
2. 写-写情况

在写-写情况下会发生脏写问题，任何一种隔离级别都不允许发生这种问题，在多个未提交事务相继对一条记录进行改动时，需要让它们排队执行，排队过程是通过加锁实现的。锁本质是内存中的结构，当一个事务想要对这条记录进行改动时，首先查看内存中有没有与这条记录关联的锁结构，如果没有，就会在内存中生成一个锁结构与之关联。

锁结构中有很多信息，最重要的是两个，一个是trx，表示这个锁结构与哪个事务关联的，另一个是ix\_waiting，表示当前事务是否在等待。并不是所有的加锁操作都需要生成对应的锁结构，有时使用“隐式锁”的方式，仍然能够起到保护记录的作用。

1. 读-写情况

在涉及到读的情况中，就会出现脏读，不可重复读，幻读问题，怎么避免这些问题呢？有两种可选的方案。

1. 读操作使用MVCC，写操作加锁

MVCC通过生成一个ReadView找到符合条件的记录版本，即在生成ReadView的时刻生成一个快照，查询语句（读操作）只能查询到该快照之前已提交事务所做的更改。对于写操作来说，肯定是对最新版本的记录进行更改，读记录的历史版本与改动记录的最新版本并不冲突，即采用MVCC时，读-写操作并不冲突。

1. 读，写操作都加锁

如果有一些业务场景不允许读取记录的旧版本，而是每次必须读取记录的最新版本，这样读取操作就需要进行加锁了。

很明显，采用MVCC方式，读-写操作并不冲突，性能更高；采用加锁方式，读写操作均需要排队执行，影响性能。

1. 一致性读

事务利用MVCC进行的读取操作称为一致性读，其他事务可以自由地对表中的记录进行改动。

1. 锁定读
2. 共享锁和独占锁

共享锁：S锁，事务读取一条记录时，需要先获取该记录的S锁，其他事务也可以获取该S锁；

独占锁：X锁，排它锁，事务要改动一条记录时，需要先获取该记录的X锁，其他事务无法获取该记录的X锁或S锁；

1. 锁定读

在读取记录前就为该记录加锁的读取方式称为锁定读，有两种特殊的SELECT语句支持锁定读：

1. 对读取的记录加S锁

SELECT … LOCK IN SHARE MODE;

1. 对读取的记录加X锁

SELECT … FOR UPDATE;

1. 写操作
2. DELETE

在B+树中定位到记录的位置，获取记录的X锁，执行delete mark操作。

1. UPDATE

定位，获取X锁。

1. INSERT

新插入的一条记录受隐式锁保护，不需要在内存中生成锁结构。

1. 多粒度锁
2. 行锁与表锁
3. 行锁

对一条记录加锁，粒度教习。

1. 表锁

对一个表加锁，粒度较粗。表锁也可以分为S锁和X锁，对于S锁，其他的事务照样可以获得该表的S锁或者表中记录的S锁，照样不能获取表和记录的X锁。对于表的X 锁，其他事务啥也干不了。

1. 意向锁
2. 意向共享锁

IS锁，当事务准备在某条记录上加S锁时需要先在表级别上加一个IS锁。

1. 意向独占锁

IX锁，当事务准备在某条记录上加X锁时，需要先在表级别上加一个IX锁。

IS和IX锁都只是在对表加S锁或者X锁时用到的，用来快读地判断当前表中是否有记录被上锁了，避免使用遍历的方式判断。当要给表加S锁时，先看看有没有IX锁，如果没有才可以加S锁。不需要检查IS锁，因为有IS锁，也可以给表加S锁。当要给表加X锁时，要判断是否有IS锁或IX锁，只有两个锁都释放了，才可以添加X锁。

1. MySQL中的行锁和表锁
2. InnoDB中的锁
3. 表级别的S锁X锁

在对某个表执行SELECT, INSERT, DELETE, UPDATE语句时，InnoDB是不会为这个表添加表级别的S锁或者X锁的。在对某个表执行一些改动表结构的DDL语句时，其他事务在对这个表并发执行查询，写入语句时是阻塞的，这是通过server层中的元数据锁（MDL）实现的。InnoDB中的表级别的S锁，X锁没有太大作用，可能会在恢复时用到。

1. 表级别的AUTO-INC锁

为表的某个列添加AUTO\_INCREMENT属性，之后插入记录时可以不指定该列的值，该列是自动递增的。该列的自动递增可以通过两个方式实现：

1. AUTO-INC锁

在执行插入语句时加一个表级别的AUTO-INC锁，为每条待插入记录的自动递增列分配递增的值，在该语句执行结束后，再把AUTO-INC锁释放掉，这样，其他事务的插入语句都会阻塞，保证一个语句中分配的递增值是连续的。这个锁的作用范围是单个插入语句。

1. 轻量级锁

获取锁，在生成该列的值后释放掉锁，不需要等到整个插入语句执行完毕。如果插入前能够确定要插入多少条记录，则采用轻量级锁，以免锁定整个表。

1. 行级锁
2. Record Lock

记录锁，仅仅锁上一条记录，分为X锁和S锁。

1. Gap Lock

MySQL在可重复读隔离级别下是可以在很大程度上解决幻读现象的，可以通过MVCC 或者加锁解决，但是在加锁时，事务在第一次执行读取操作时，那些幻影记录是不存在的，我们无法给这些记录加上正常的记录锁。gap锁就能够解决这种问题，当给某个记录加上gap锁时，表示不允许别的事务在该记录前面的间隙中插入新纪录，这能能够防止插入幻影记录。

1. next-key lock

既锁住某条记录，又阻止其他事务在该记录的前面的间隙中插入新纪录，这种记录称为next-key锁，即正经记录锁和gap锁的合体。

1. Insert Intention Lock

一个事务在插入一条记录时，需要判断插入位置是否已经被其他的事务加了gap锁，如果有的话需要等待，等待时也要在内存中生成一个锁结构，称为插入意向锁，没有太大作用。

1. 隐式锁

一般执行INSERT语句是不需要在内存中生成锁结构的，但是没有锁的记录很容易被其他事务获取X锁或者S锁，这样就会产生脏读或者脏写问题。这时可以通过事务id来解决这个问题，相当于隐式锁，回先帮助当前事务生成一个锁结构，再为自己生成一个锁结构，并进入等待状态。

对于聚簇索引记录来说，trx\_id隐藏列记录着该记录的事务id，当其他事务想要对该记录添加S锁或者X锁时，会查看该事务id是否已提交，如果已提交则正常读取；否则帮助该事务创建一个X锁的锁结构，并未自己也创建一个锁结构，进行等待状态。

对于二级索引记录来说，本身没有trx\_id列，Page Header中有一个属性表示对该页面做改动的最大的事务id，如果该属性小于当前最小的活跃事务id，则说明做修改的事务都提交了。

1. 锁的内存结构

不可能对每一条记录都创建锁结构，同一事务的可以放到一个锁结构中，被加锁的记录在同一个页面中的可以放到一个锁结构中，加锁的类型是一样的可以放到一个锁结构中。

1. 查看事务加锁情况
2. information\_schema数据库

在information\_schema系统数据库中，有几个与事务和锁相关的表：

1. INNODB\_TRX

存储了InnoDB当前正在执行的事务信息。

1. INNODB\_LOCKS

记录了一些锁信息，如某个事务为了某个锁而等待则记录该锁信息，如果某个锁阻塞了别的事务，则记录该锁信息。

1. SHOW ENGINE INNODB STATUS

获取当前系统中各个事务的加锁情况。

1. 死锁

当死锁发生时，InnoDB会回滚一个事务释放掉该事务所获取的锁，我们需要找出那些发生死锁的语句，通过优化语句来改变加锁顺序，或者建立合适的索引来改变加锁过程，需要通过死锁日志来分析，定位死锁的语句。