mysql的锁是面试中很高频问题，也是我们在日常开发中经常会遇到但是我们并没有注意到的地方。我把我自己理解的锁通过本篇博文分享出来，由于锁需要结合事务来理解，本文只介绍锁的基本概念，同样为了理解事务会更加深刻，先介绍了InnoDB的一些基础概念，也是记录自己的学习，欢迎大家一起探讨交流。

### 锁的分类：

* 按照锁的粒度来分
  + 全局锁: 锁的是整个database，类比一个库为一栋大楼，那此时就是锁的整栋楼的大门
  + 表级锁： 锁的是某个table，类比一个表为大楼的某一层，此时锁的就是某一层整层
  + 行级锁：锁的是某一行数据，类比每行锁的是某一层的某一间房间，此时锁的是某一个房间
* 表锁和行锁的区别
  + 表级锁，开销大，加锁快，不会出现死锁。锁的粒度大。并发度低。
  + 行级锁，开销小，加锁慢，会出现死锁，锁的粒度小，并发度高。

### 表级锁：

mysql的表级锁有两种：元数据锁和表锁。

表锁的两种形式：

* 表共享读锁
* 表排它写锁
  + 手动加表锁
  + lock table tableName read;
  + 查看表锁情况
  + show open tables;
  + 删除表锁
  + unlock tables;

元数据锁：

* 5.5版本中引入了MDL，对一个表数据进行增删改的时候，加MDL读锁；要对表结构进行修改的时候，加MDL写锁。

### 行级锁

mysql的行级锁是有存储引擎实现的，mysql现在默认的数据引擎为Innodb。本文主要介绍InnoDB的行锁；

InnoDB的行锁是给索引项加锁实现的，也就意味着只有使用索引检索的数据才能使用行锁，否则将使用表锁。

#### 按照范围来说

* 行锁：锁定表中的某一条记录。
* 间隙锁：
  + 锁住索引记录中间的值
  + 锁住第一个索引记录前面的值或者最后一个索引后面的值

#### 按照功能来说

* 共享锁，也叫做S锁：允许一个事务去读一行数据，阻止其他事务添加排它锁，允许继续添加共享锁读
* 排它锁，也叫做X锁：允许获得排它锁的事务更新数据，阻止其他事务添加读共享锁和添加排它锁写

对于InnoDB来说，会自动给增删改语句添加排它锁，X锁。对于普通的查询语句不会添加任何锁。

#### 意向锁

InnoDB同样也实现了表级锁，也就是意向锁。意向锁是mysql内部使用的，不需要用户去干预。

* 意向共享锁，IS锁：事务打算给数据行加共享锁，事务在给一个数据行添加共享锁前必须获取该表的IS锁。
* 意向排它锁，IX锁：事务打算给数据航加排它锁，事务在给一个数据行添加排它锁前必须获取该表的IX锁。

意向和行锁可以共存，意向锁的作用是为了提升全表更新数据时的性能提升，否则在更新全表时要检索哪些数据行上有行锁。

#### 间隙锁

顾名思义，主要是在记录之间添加锁，不允许往间隙插入数据。比如id为 2 4，那此时使用间隙锁就会锁2 3 4 这三个，稍后在介绍事务的时候也会再次介绍间隙锁，间隙锁的主要作用就是为了解决幻读问题。此处先了解一下。

#### 死锁

mysql的死锁和我们代码中死锁理论是一样的，不同的是，mysql指的是两个不同的连接互相等待对方释放锁，才能释放自己持有的资源，所以造成了死锁。mysql中也有对死锁的优化。我们稍后再具体说。

接下来我们开始介绍事务，上面只是简单介绍了一下锁的基本概念，锁还有一部分内容需要结合事务来理解，所以稍后还有锁的介绍。

在我们介绍事务之前，我们先聊一下InnoDB的架构，事务中的一些部分会涉及到这部分的内容。

### InnoDB的磁盘文件

#### InnoDB的磁盘文件

* 系统表空间
  + 系统表空是一个共享的表空间
  + 系统表空间包含数据字典、doule write buffer、change buffer、undo log的存储区域，包含用户在系统表创建的表结构和索引数据
* 用户表空间
  + 设置参数 innodb*file*per\_table ，用户就可以为每个基于InnoDB引擎的表创建一个独立的用户表空间，也就是.ibd文件。
  + 存储该表的数据、索引等信息。

#### InnoDB内存结构

* buff pool 缓冲池
  + 数据是存储于磁盘的，由于cpu速度和磁盘速度的差别，所以使用缓冲池提高整体性能。
  + 通过innodb*buffer*pool\_size可以设置缓冲池的大小，缓冲池的大小对性能也是有影响的。
  + 缓冲池中缓冲的数据类型：
    - 索引页
    - 数据页
    - 存储引擎工作时，需要以页为单位将磁盘数据加载到内存中，数据页和索引页是页类型中最重要的两种类型
    - undo页:实现了mysql多版本的快照，可以理解为版本链。mvcc和回滚操作都涉及到了undo日志。
    - insert buffer：提高了对于非聚簇索引的插入性能
    - 自适应哈希索引
    - InnoDB存储的锁信息
    - 数据字典信息

#### 内存数据落盘

InnoDB数据落盘有图可以看出来是通过两种方式来实现的

* 脏页数据落盘
* 预写redo日志

通过两种方式来落盘，也可以理解为持久化到磁盘上。是为了保证数库发生突然宕机，造成数据丢失。

脏页落盘会产生IO并且是随机写入，耗时比较长。频繁进行磁盘IO对性能损耗是非常大。并且数据的安全性得不到保障。如果在脏页数据还没来得及落盘或者落盘过程出现宕机，那么数据就会丢失。

鉴于以上情况，mysql用双保险完成数据的安全性，脏页落盘是一种，另一种就是预写redo 日志，首先我们要知道redo 持久化到磁盘是顺序写入，顺序写入的速度要比随机写入要快，此时有朋友就会问，那脏页落盘为什么不采用顺序写入呢但？

顺序写入速度快的同时是会产生磁盘碎片的，磁盘碎片会大大浪费磁盘资源。

redo 日志持久化的时机是在事务提交时写入到磁盘的redo file中，此时脏页数据并不一定完成了落盘，脏页落盘是由checkPoint检查点机制控制的，我们这里不展开多说。

数据库发生宕机的情况：

* 脏页数据未落盘，事务未提交，此时产生了数据丢失，我们都知道如果事务未提交换个角度来讲这些数据丢失是正常的。
* 脏页数据未落盘，事务已提交，此时redo log file已经有了数据，那么重启的时候mysql就会从redo log file中进行数据恢复。

有的朋友还会说，那redo log file的数据岂不是无限大？

ib*logfile0，ib*logfile1 这是rodo log 在我们磁盘上的命名，可以看到有两个文件，采用的循环写入的方式，如果1满了就写入2,2满了写入1，这样循环。

redo 日志持久化到磁盘也是可以配置的，通过InnoDB的innodb*push*log*at*trx\_commit来设置

1. 属性值为0时，事务提交，不会对redo进行写入操作，等待主线程按时写入；
2. 属性值为1时，事务提交，将数据写入磁盘，确保不会出现数据丢失；
3. 属性值为2时，事务提交，将数据写入系统缓存，让文件系统自己判断什么时候写入磁盘。

默认值为1，一般也建议设置为1，会保证数据的安全性，并且只有为1的时候才会保证事务的一致性。