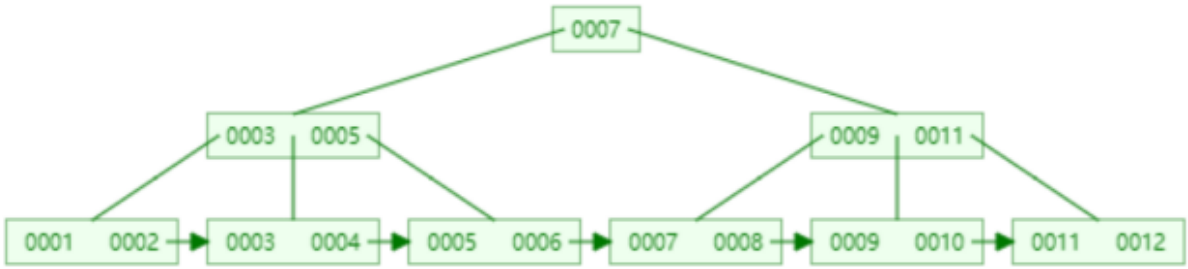
一、加锁的目的是什么？

在了解数据库锁之前，首先我们必须要明白加锁的是为了解决什么问题，如果你还不清楚的话，那么从现在开始你就知道了，对数据加锁是为了解决事务的隔离性问题，让事务之间相互不影响，每个事务进行操作的时候都必须先对数据加上一把锁，防止其他事务同时操作数据。如果你想一个人静一静，不被别人打扰，那么请在你的房门上加上一把锁。

二、锁是基于什么实现的?

为了后面大家后面对锁理解的更透彻，所以必须要先解决这个问题，现实生活中家里的锁是锁在门上的，而数据库里面的锁是基于索引实现的，在Innodb中我们的锁都是作用在索引上面的，当我们的SQL命中索引时，那么锁住的就是命中条件内的索引节点(行锁)，如果没有命中索引的话，那我们锁的就是整个索引树（表锁），如下图一下锁住的是整棵树还是某几个节点，完全取决于你的条件是否有命中到对应的索引节点。

innodb索引结构图(B+ tree):



三、锁的分类。

数据库里有的锁有很多种，为了方面理解，所以我根据其相关性"人为"的对锁进行了一个分类，分别如下:

基于锁的属性分类：**共享锁、排他锁。**

基于锁的粒度分类：**表锁、行锁(记录锁、间隙锁、临键锁)。**

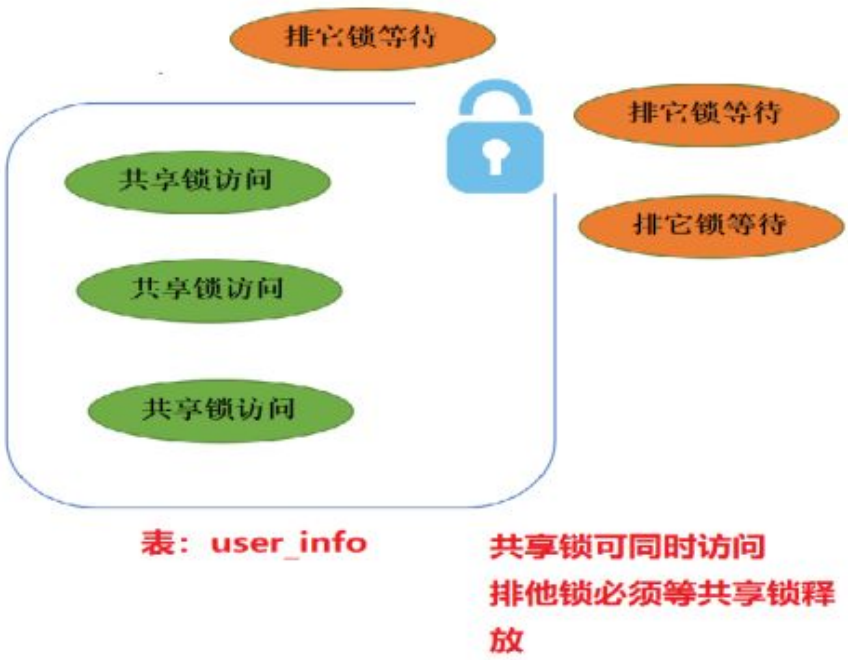
基于锁的状态分类：**意向共享锁、意向排它锁。**

3.1、属性锁

共享锁(Share Lock)

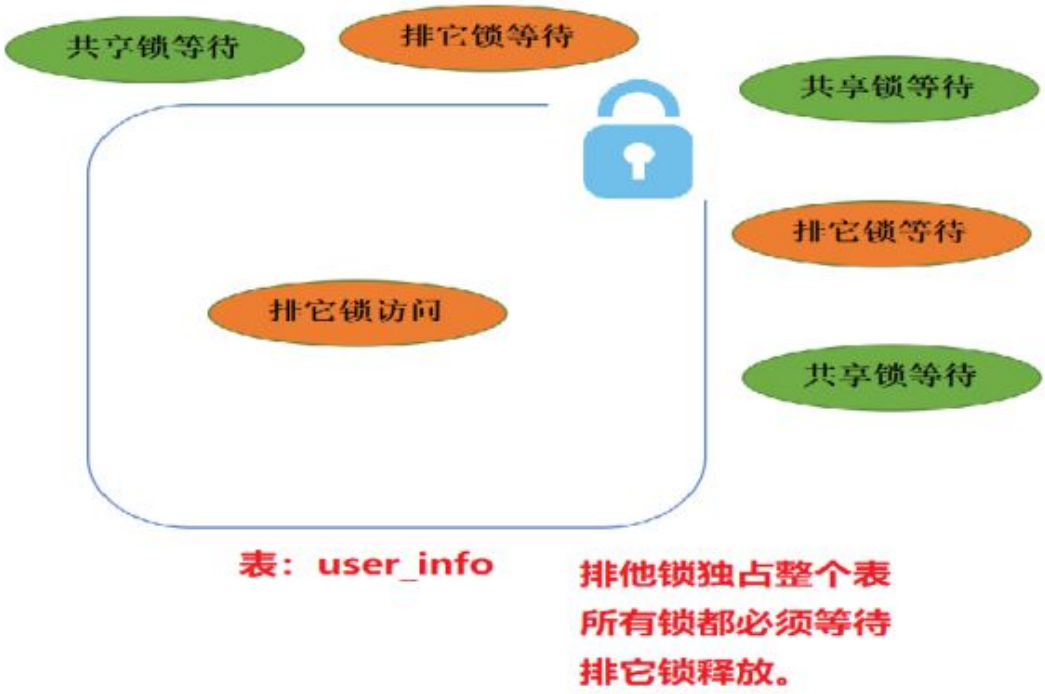
共享锁又称读锁，简称S锁。当一个事务对数据加上读锁之后，其他事务只能对该数据加读锁，而无法对数据加写锁，直到所有的读锁释放之后其他事务才能对其进行加写锁。 加了共享锁之后，无法再加

排它锁，这也就可以避免读取数据的时候会被其它事务修改，从而导致重复读问题。



排他锁（eXclusive Lock）

排他锁又称写锁，简称X锁；当一个事务对数据加上写锁之后，其他事务将不能再为数据加任何锁，直到该锁释放之后，其他事务才能对数据进行加锁。加了排他锁之后，其它事务就无法再对数进行读取和修改，所以也就避免了脏写和脏读的问题。

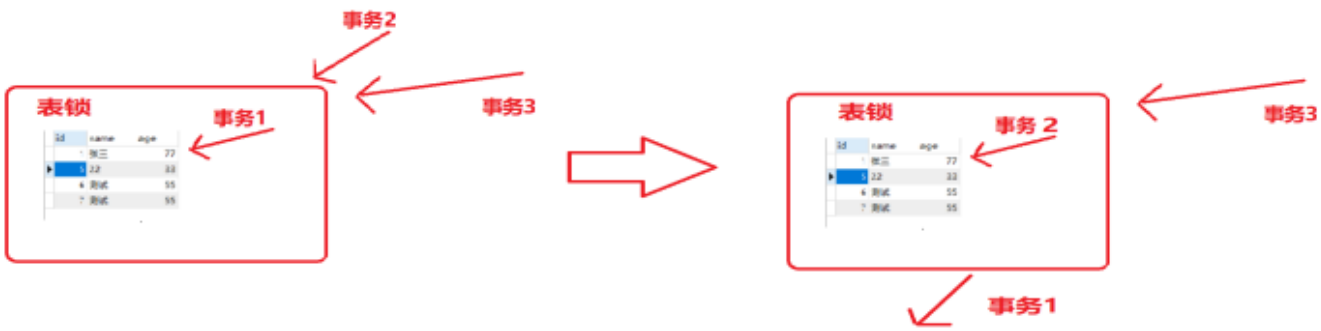


3.2、粒度锁

表锁

表锁是指上锁的时候锁住的是整个表，当下一个事务访问该表的时候，必须等前一个事务释放了锁才能进行对表进行访问；

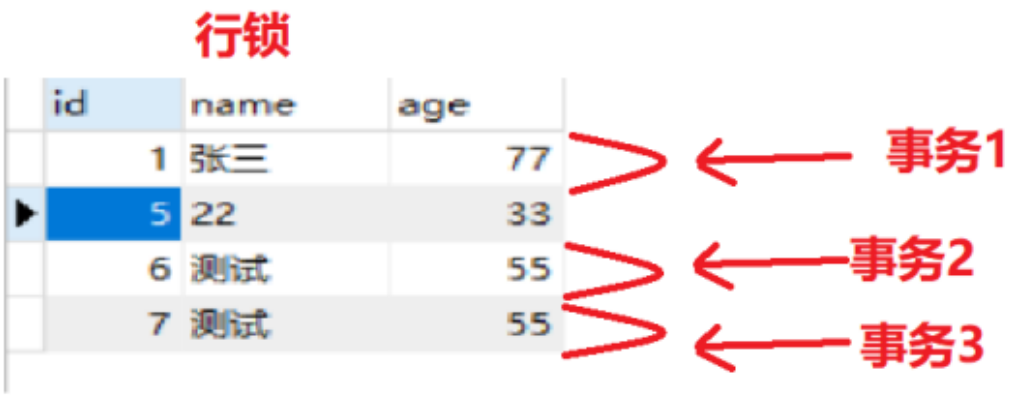
特点： 粒度大，加锁简单，容易冲突；



行锁

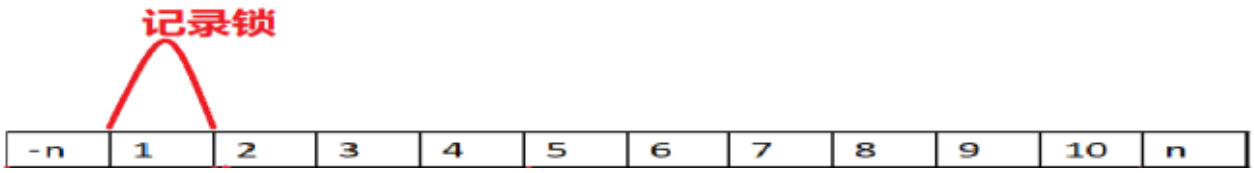
行锁是对所有行级别锁的一个统称，比如下面说的记录锁、间隙锁、临键锁都是属于行锁， 行锁是指加锁的时候锁住的是表的某一行或多行记录，多个事务访问同一张表时，只有被锁住的记录不能访问，其他的记录可正常访问；

特点：粒度小，加锁比表锁麻烦，不容易冲突，相比表锁支持的并发要高；



记录锁(Record Lock)

记录锁属于行锁中的一种，记录锁的范围只是表中的某一条记录，记录锁是说事务在加锁后锁住的只是表的某一条记录。



触发条件：精准条件命中，并且命中索引；

例如：update user_info set name='张三' where id=1 ,这里的id是索引。

记录锁的作用：加了记录锁之后数据可以避免数据在查询的时候被修改的重复读问题，也避免了在修改的事务未提交前被其他事务读取的脏读问题。

间隙锁(Gap Lock)

间隙锁属于行锁中的一种，间隙锁是在事务加锁后其锁住的是表记录的某一个区间，当表的相邻ID之间出现空隙则会形成一个区间，遵循左开右闭原则。

比如下面的表里面的数据ID 为 1,4,5,7,10 ,那么会形成以下几个间隙区间，-n-1区间，1-4区间，7-10区间，10-n区间（ -n代表负无穷大，n代表正无穷大）

id	name	age
1	a	18
4	b	18
5	c	18
7	d	18
10	e	18

触发条件：范围查询，查询条件必须命中索引、间隙锁只会出现在REPEATABLE_READ（重复读)的事务级别中。

例如：对应上图的表执行select * from user_info where id>1 and id<4(这里的id是唯一索引），这个SQL查询不到对应的记录，那么此时会使用间隙锁。

间隙锁作用：防止幻读问题，事务并发的时候，如果没有间隙锁，就会发生如下图的问题，在同一个事务里，A事务的两次查询出的结果会不一样。

A事务	B事务
begin	/
select * from user_info where id>1 and id<4;	/
/	begin
/	insert into user_info(id,name,age)values(2,zhangsan,18)
/	commit
select * from user_info where id>1 and id<4;	/
commit	/

临键锁(Next-Key Lock)

临键锁也属于行锁的一种，并且它是INNODB的行锁默认算法，总结来说它就是记录锁和间隙锁的组合，临键锁会把查询出来的记录锁住，同时也会把该范围查询内的所有间隙空间也会锁住，再之它会把相邻的下一个区间也会锁住。

例如：下面表的数据执行 select * from user_info where id>1 and id<=13 for update ;

会锁住ID为5,10的记录；同时会锁住，1至5,5至10,10至15的区间。

id	name	age
1	a	18
5	b	18
10	c	18
15	d	18

触发条件：范围查询，条件命中了索引。

临键锁的作用：结合记录锁和间隙锁的特性，临键锁避免了在范围查询时出现脏读、重复读、幻读问题。加了临键锁之后，在范围区间内数据不允许被修改和插入。

3.3、状态锁

状态锁包括意向共享锁和意向排它锁，把他们区分为状态锁的一个核心逻辑，是因为这两个锁都是描述是否可以对某一个表进行加表锁的状态。

意向锁的解释：当一个事务试图对**整个表**进行加锁（共享锁或排它锁）之前，首先需要获得对应类型的意向锁（意向共享锁或意向共享锁）

意向共享锁: 当一个事务试图对整个表进行加共享锁之前，首先需要获得这个表的意向共享锁。

意向排他锁: 当一个事务试图对整个表进行加排它锁之前，首先需要获得这个表的意向排它锁。

为什么我们需要意向锁？

意向锁光从概念上可能有点难理解，所以我们有必要从一个案例来分析其作用，这里首先我们先要有一个概念那就是innodb加锁的方式是基于索引，并且加锁粒度是行锁，然后我们来看下面的案例。

第一步：

事务A对user_info表执行一个SQL:update user_info set name =”张三” where id=6 加锁情况如下图;

第二步：

与此同时数据库又接收到事务B修改数据的请求：SQL: update user_info set name =”李四”；

1、因为事务B是对整个表进行修改操作，那么此SQL是需要对整个表进行加排它锁的（update加锁类型为排他锁）；
2、我们首先做的第一件事是先检查这个表有没有被别的事务锁住，只要有事务对表里的任何一行数据加了共享锁或排他锁我们就无法对整个表加锁（**排他锁不能与任何属性的锁兼容**）。
3、因为INNODB锁的机制是基于行锁，那么这个时候我们会对整个索引每个节点一个个检查，我们需要检查每个节点是否被别的事务加了共享锁或排它锁。
4、最后检查到索引ID为6的节点被事务A锁住了，最后导致事务B只能等待事务A锁的释放才能进行加锁操作。

思考：
在A事务的操作过程中，后面的每个需要对user_info加持表锁的事务都需要遍历整个索引树才能知道自己是否能够进行加锁，这种方式是不是太浪费时间和损耗数据库性能了？
所以就有了意向锁的概念：如果当事务A加锁成功之后就设置一个状态告诉后面的人，已经有人对表里的行加了一个排他锁了，你们不能对整个表加共享锁或排它锁了，那么后面需要对整个表加锁的人只需要获取这个状态就知道自己是不是可以对表加锁，避免了对整个索引树的每个节点扫描是否加锁，而这个状态就是我们的意向锁。

四、在测试锁的实操过程中需要注意的问题。

看了这么多理论，相信很多人都会去实际操作一下，一是验证逻辑，而是加深自己的理解，为了避免大家少踩坑，在这里提醒几个需要注意的地方。

1、关闭自动提交事务功能，自己手动begin commit rollback。

```
show variables like 'autocommit'
Set autocommit = 0
```

```
begin

select * from tb_user

commit
```

2、查看当前会话隔离级别是否为REPEATABLE-READ（一般默认都是此级别）

```
SELECT @@tx_isolation
```

3、最重要的一点，查询数据的时候要使用当前读（因为Mysql 有MVCC的机制所以很多情况下都不会进行加锁，使用当前读就不会使用MVCC）比如使用下面这个 for update 就是使用当前读。

```
1
2  BEGIN
3
4  select * from tb_user where id>3 and id<10 for update
5
6  COMMIT
```