**MySQL**

# MyISAM和InnoDB的区别

1. MyISAM只支持表锁，InfoDB支持表锁和行锁。
2. MyISAM不支持事务，InfoDB支持事务。



# MyISAM

## 共享读锁

1. 创建一张表

|  |
| --- |
| CREATE TABLE `testisam` (  `id` int(11) DEFAULT NULL  ) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4; |

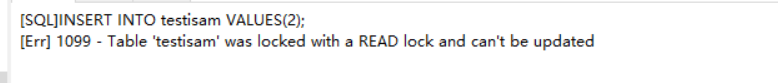
1. 插入数据

|  |
| --- |
| INSERT INTO testisam VALUES(2); |

1. 开启行锁

|  |
| --- |
| LOCK TABLE testisam READ; |

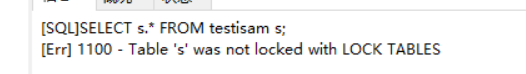
1. 再次执行第二步中的插入语句，结果如下：



1. 加上别名

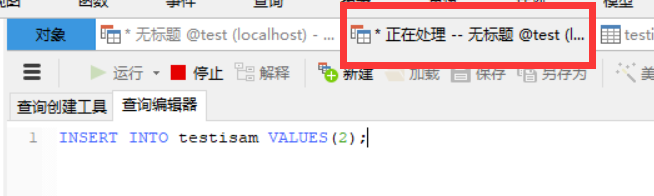
|  |
| --- |
| SELECT s.\* FROM testisam s; |

查询报错:

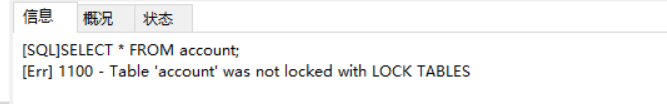


1. 打开另一个会话，继续进行插入语句，结果如下:

处于等待状态



1. 在开启行锁的相同的会话中，查询其他的表，结果如下:



1. 解锁

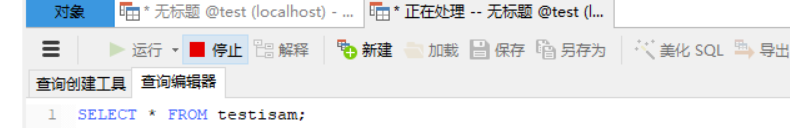


## 写锁

1. 开启读锁

|  |
| --- |
| LOCK TABLE testisam WRITE; |

1. 执行插入、更新、删除、查询语句都是可以的。
2. 查询其他表也是不能成功的（和开启读锁同一个会话）。
3. 在另一个会话中，查询testisam 中的数据，结果：**等待中**



## 总结锁

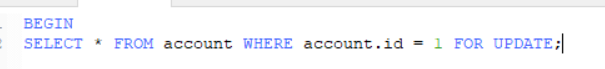


# InnoDB

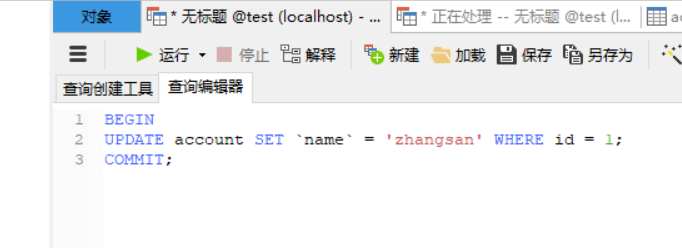
只支持B-tree索引

## 行锁

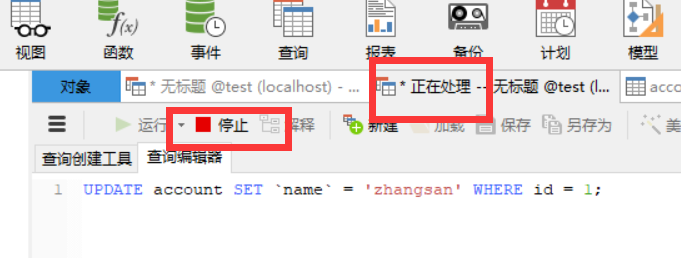
1. 开启行锁



**Update语句会自动添加行锁**



1. 在另一个会话中修改数据，id一样，结果**:在等待，因为开启了行锁，在将上面的事务提交之后，该条数据自动修改成功。**



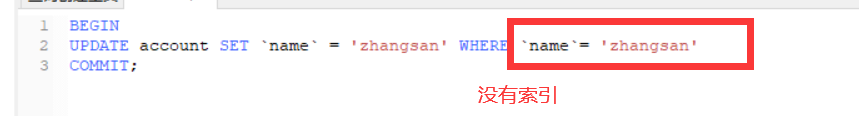
**超时之后，自动取消**



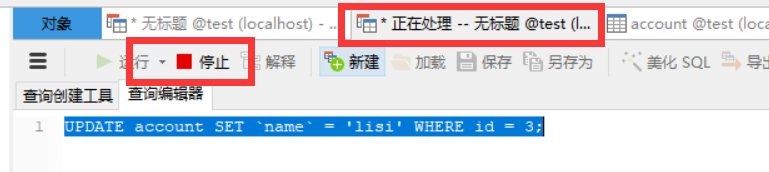
1. 在另一个会话中修改数据，但是id不一样，结果是可以修改成功，因为和锁住的数据的id不一样。

---------------------------------------------------------------------------------------------------------------------------------

1. 开启事务，执行一条修改语句，条件涉及的字段是没有索引的

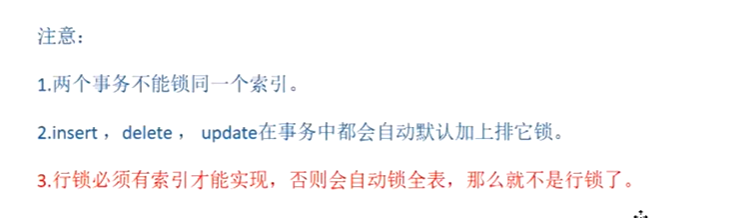


1. 在另一个会话中修改数据，与上个语句的条件不一样，结果:数据不一样，为啥还修改不成功呢？答案：在上一个会话中SQL语句涉及的字段是没有索引的，那么修改的时候会锁住所有的数据，所以这里修改的时候会等待上一个事务提交。

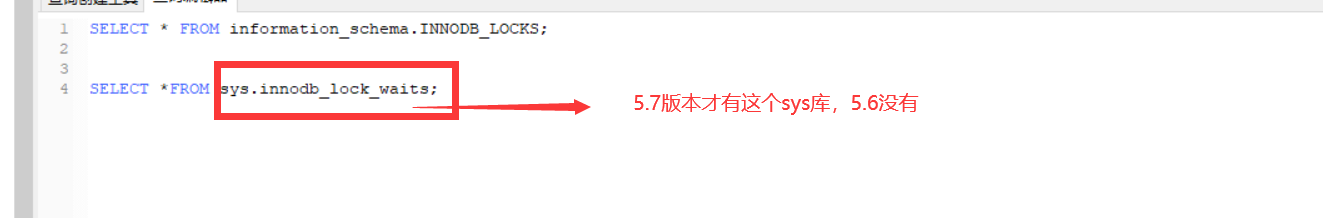


## 总结（行锁）

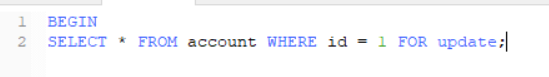
两条SQL语句，id一样，但是只能有一条才能修改成功，如果加上版本号，一条数据修改成功之后，版本号+1，另一条SQL语句再执行不会成功，因为版本号已经改变了，这就是乐观锁。

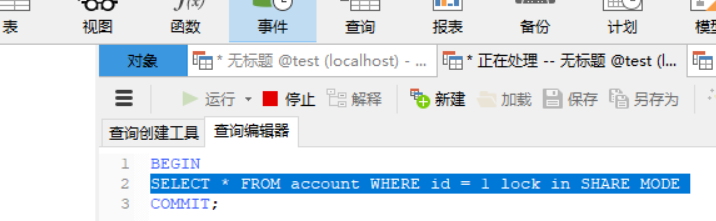


## 查询锁的信息



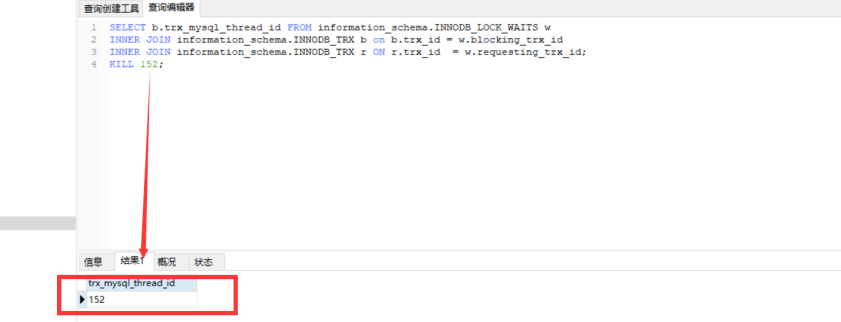
## 解决锁的问题，在开发过程中可能会遇到



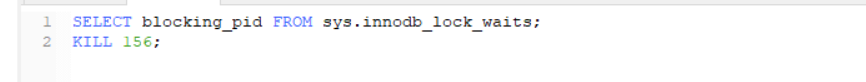


执行这个第二提交SQL语句，在等待中，解决方法如下(5.6版本):

|  |
| --- |
| SELECT b.trx\_mysql\_thread\_id FROM information\_schema.INNODB\_LOCK\_WAITS w  INNER JOIN information\_schema.INNODB\_TRX b on b.trx\_id = w.blocking\_trx\_id  INNER JOIN information\_schema.INNODB\_TRX r ON r.trx\_id = w.requesting\_trx\_id; |



### 5.7版本解决锁等待的问题



# 事务

## 什么是事务

事务是指是程序中一系列严密的逻辑操作，而且所有操作必须全部成功完成，否则在每个操作中所作的所有更改都会被撤消。可以通俗理解为：就是把多件事情当做一件事情来处理，好比大家同在一条船上，要活一起活，要完一起完 。

## 事务的特性

* **原子性**（Atomicity）**：**操作这些指令时，要么全部执行成功，要么全部不执行。只要其中一个指令执行失败，所有的指令都执行失败，数据进行回滚，回到执行指令前的数据状态。
* **一致性**（Consistency）**：**事务的执行使数据从一个状态转换为另一个状态，但是对于整个数据的完整性保持稳定。
* **隔离性**（Isolation）**：**隔离性是当多个用户并发访问数据库时，比如操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离。
* **持久性**（Durability）**：**当事务正确完成后，它对于数据的改变是永久性的。

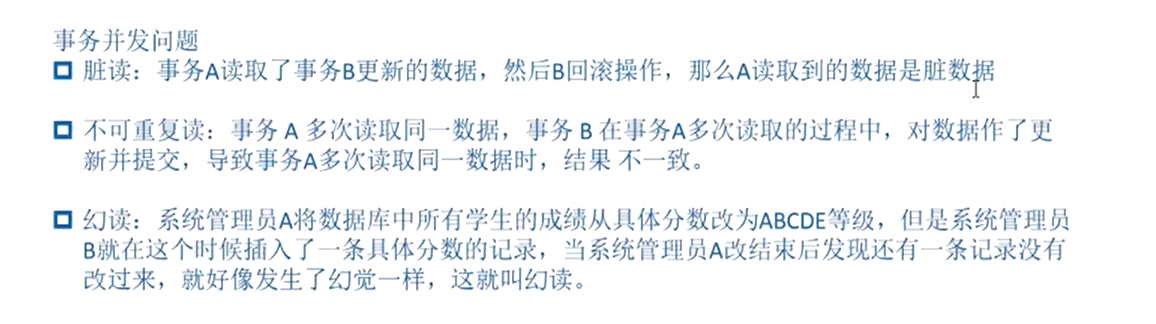
## 隔离性

未提交读(脏读)

已提交读(不可重复读)

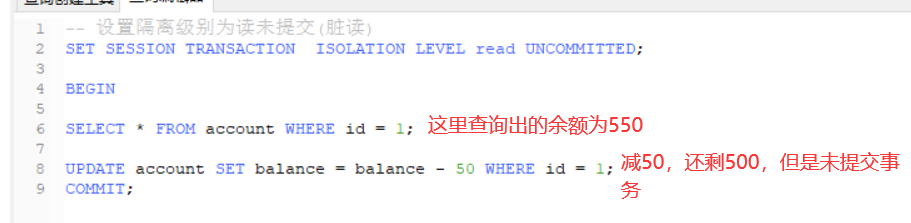
可重复读

序列化



## 通过案例来演示事务的隔离性

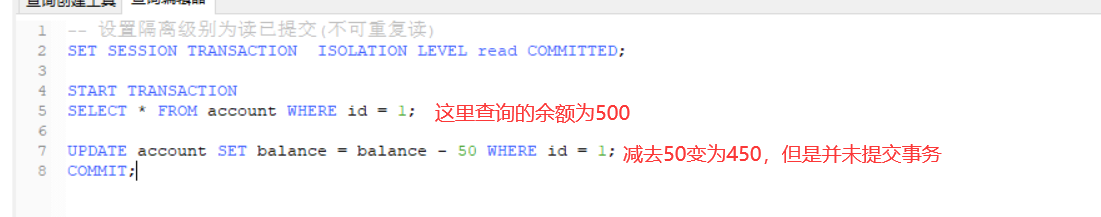
### 读未提交

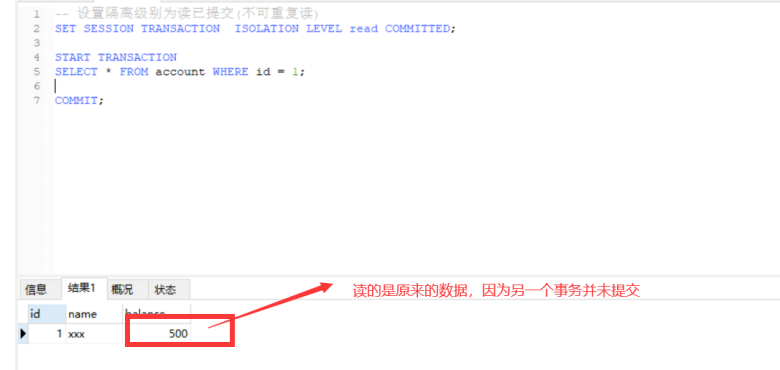




### 读已提交（不可重复读）

|  |
| --- |
| -- 设置隔离级别为读已提交(不可重复读)  SET SESSION TRANSACTION ISOLATION LEVEL read COMMITTED;  START TRANSACTION  SELECT \* FROM account WHERE id = 1;  UPDATE account SET balance = balance - 50 WHERE id = 1;  COMMIT; |



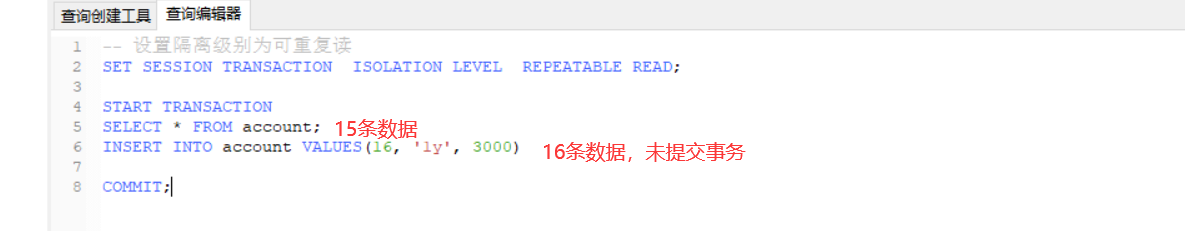


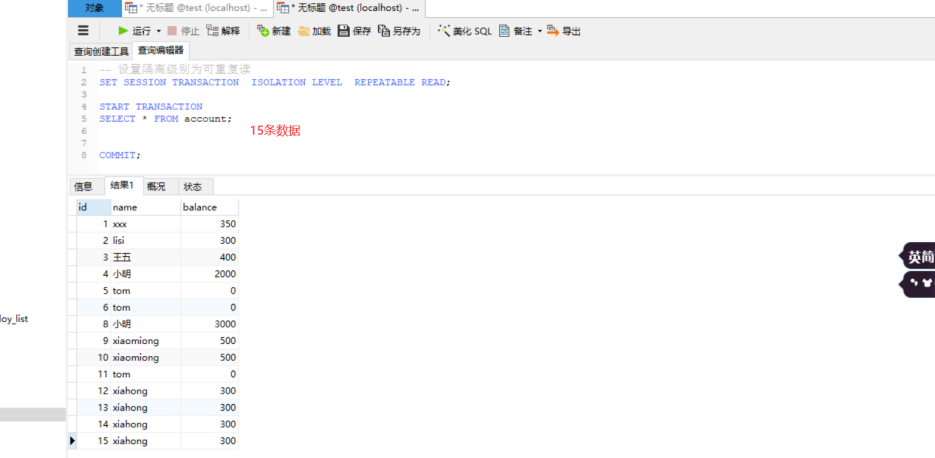
将第一个事务提交，余额变为450，第二个会话再去查询，结果也变为了450，第二个会话查询的两次的结果并不一样，这种情况叫做不可重复读。

### 可重复读

解决了不可重复读的问题

**存在的问题:幻读**



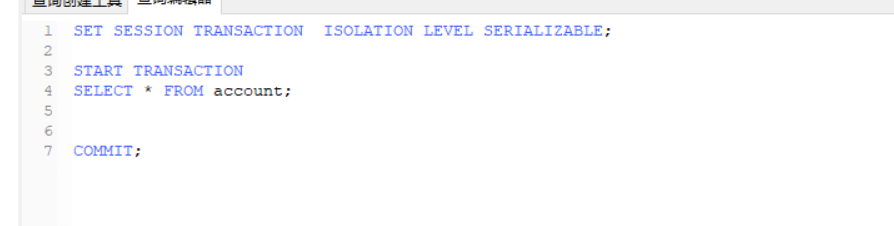


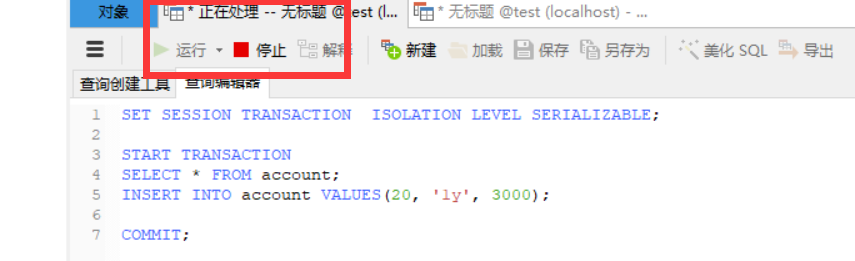
将第一个会话中的事务提交

再在第二会话中查询，结果如下:结果还是15条数据，然后再插入一条id为16的数据，却插入不进去，因为id=16的数据已经存在了，这种情况叫做幻读。

### 可序列化

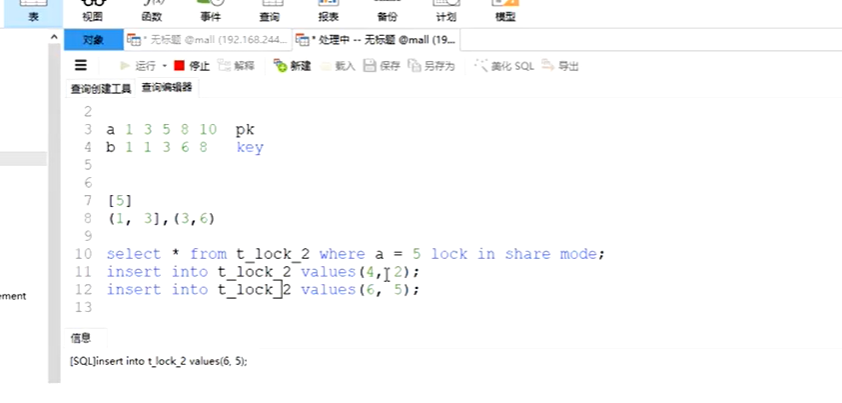
事务串行执行，一个一个执行，a事务在执行，b执行必须等到a事务提交之后才能执行。





# 间隙锁

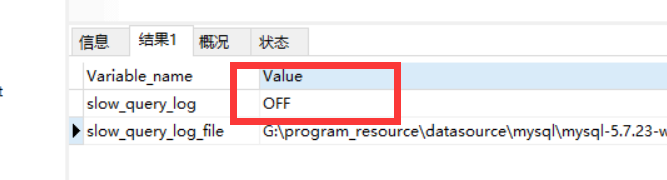
a为主键，b为普通的索引



# 慢查询

show VARIABLES LIKE '%slow\_query\_log%'

默认是关闭的



SET GLOBAL slow\_query\_log = 1;// 开启慢查询

SET GLOBAL long\_query\_time=0;// 查过0秒的查询都会记录慢查询日志文件中

可能修改查询时间并没有生效，只要将会话关闭再重新打开就生效了。