数据库的隔离级别

# 重要问题

## 数据库事务的特性ACID；

## 数据库事务的隔离级别及对应现象；

## MySQL的InnoDB与Myisam的区别。

# 数据库事务(Transaction,TX)介绍

## 事物定义

事务：**一个最小的不可再分的工作单元**；通常一个事务对应一个完整的业务。

(例如银行账户转账业务，该业务就是一个最小的工作单元)

一个完整的业务需要批量的**DML(insert、update、delete)语句**共同联合完成；

**事务只和DML语句有关，或者说DML语句才有事务。与select无关。**

数据操纵语言（Data Manipulation Language, DML）

**manipulation** 英 [mə.nɪpjə'leɪʃ(ə)n] 美 [mə.nɪpjʊ'leɪʃ(ə)n]

n. **操纵; 控制**; （熟练的） 操作; （对账目等的） 伪造;

## 事务的ACID属性

### 原子性：Atomicity

### 一致性：Consistency

### 隔离性：Isolation

### 持久性：Durability

* 原子性(A)：事务是最小单位，不可再分；
* 一致性(C)：事务要求所有的**DML语句**操作的时候，必须保证**同时成功或者同时失败**；
* 隔离性(I)：事务A和事务B之间具有隔离性，具有4个隔离级别。
* 持久性(D)：是事务的保证，**事务终结的标志**(内存的数据持久到硬盘文件中)。

## 举例

事务最经典的例子就是**银行转账**。

# MySQL事务的4种隔离级别

## 事务之间必须具有隔离性Isolation，隔离性具有四个级别：

A. 读未提交Read-Uncommitted

B. 读已提交 Read-Committed

C. 可重复读 Repeatable-Read

D. 可串行化 Serializable

### 读未提交Read-Uncommitted(实际很少使用)

事务A可以读取到事务B未提交的数据，读取到的数据叫做“脏数据”；

这种隔离级别**最低**，一般是在理论上存在，数据库隔离级别一般都高于该级别。

### 读已提交 Read-Committed

事务A只能读取到事务B已提交的数据，避免了“脏数据”；但是无法避免“不可重复读”；大多数数据库的默认隔离级别，如**Oracle数据库的默认隔离级别就是读已提交。**

### 可重复读 Repeatable-Read

事务A不仅读取不到事务B未提交的数据，已经提交的数据也读取不到，一直保持当前事物一开始读取的数据。避免了“脏读”和“不可重复读”；实现了“可重复读”。但是，仍会导致“幻读”。

MySQL数据库的**默认隔离级别就是可重复读**。

### 可串行化 Serializable(很少使用)

事务A在操作数据库时，事务B只能排队等待；这种隔离级别很少使用，吞吐量太低，用户体验差；这种级别可以避免“幻读”，**每一次读取的都是数据库中真实存在数据**，事务A与事务B串行，而不并发。

**select 语句不受影响，update、insert、delete同一时间只能有一个事物操作，其他事务排队等待。**

## 4种现象：

### 更新丢失：

两个事务A和B同时操作相同数据，后提交的事务会覆盖先提交的事务的处理结果，通过**乐观锁**就可以解决。

### 脏读 Dirty Read：

事务A读取了事务B已经修改但是**尚未提交的数据**；如果事务B回滚，则事务A读取的数据无效，不符合一致性。

### 不可重复读NonRepeatable Read：

事务A读取到了事务B已经提交的修改数据，不符合隔离性。

对应的“可重复读”:事务A无法读取其他事务提交的数据，只能读取事务A开始之前的数据。

### 幻读 Phantom Read：强调数据有新增加。

事务A读取到了事务B提交的**新增数据**，不符合隔离性。

## 隔离级别与一致性关系



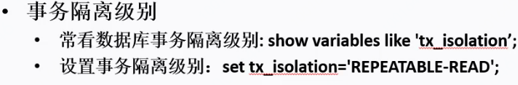
读未提交==>脏读；

读已提交==>不可重复读；

可重复读==>幻读，读到新增的数据。



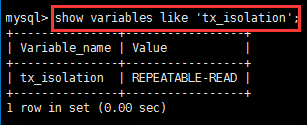
# MySQL查看与设置事务隔离级别



## 查看隔离级别

查看MySQL数据库隔离级别命令：

**show variables like 'tx\_isolation';**



**MySQL数据库事务的默认隔离级别：REPEATABLE-READ可重复读**

**其他大部分数据库的事务默认隔离级别是Read-Committed读已提交。**

## 设置事务隔离级别

### 方法1：配置文件my.ini修改：

•可以在my.ini文件中使用transaction-isolation选项来设置服务器的缺省事务隔离级别。

四个选项值：<连词符号>

**– READ-UNCOMMITTED**

**– READ-COMMITTED**

**– REPEATABLE-READ**

**– SERIALIZABLE**

• 例如：

[mysqld]

**transaction-isolation = READ-COMMITTED**

### 动态设置：通过命令动态设置隔离级别

• 隔离级别也可以在运行的服务器中动态设置，应使用**SET TRANSACTION ISOLATION LEVEL**语句。

• 其语法模式为：

**SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL <isolation-level>**

其中的<isolation-level>可以是：<不加连词符号>

**– READ UNCOMMITTED**

**– READ COMMITTED**

**– REPEATABLE READ**

**– SERIALIZABLE**

• 例如：设置会话隔离级别：

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

缺省情况设置的是Session的隔离级别。

设置**Session 会话**的事务隔离级别也可以通过以下命令实现。

set tx\_isolation='**Read-Uncommitted**';

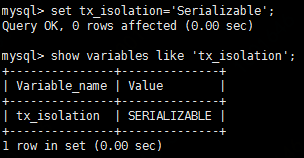
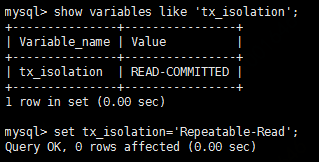
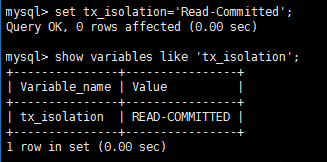
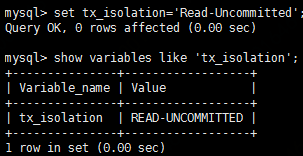
set tx\_isolation='**Read-Committed**';

set tx\_isolation='**Repeatable-Read**';

set tx\_isolation=**'Serializable'**;

命令：

**set tx\_isolation='REPEATABLE-READ';**



## 隔离级别的作用范围

事务隔离级别的作用范围分为两种：

– **全局级**GLOBAL：对所有的会话有效；

– **会话级**SESSION：只对当前的会话有效；

• 例如，设置**会话级**隔离级别为READ COMMITTED ：

mysql> SET TRANSACTION ISOLATION LEVEL READ COMMITTED；

或：mysql> SET **SESSION** TRANSACTION ISOLATION LEVEL READ COMMITTED；

• 设置全局级隔离级别为READ COMMITTED ：

mysql> SET **GLOBAL** TRANSACTION ISOLATION LEVEL READ COMMITTED；

# MySQL数据库的行锁

## 锁的基本概念

锁：是计算机协调**多个进程或线程**并发访问某一资源的机制。

锁的分类：乐观锁和悲观锁。

操作：读锁和写锁(都属于悲观锁)

粒度：**表锁**和**行锁**。**Myisam是表锁，而InnoDB是行锁。**

## 行锁：

开销大；加锁慢；会出现死锁；锁定粒度小；并发度高；偏向InnoDB存储引擎；发生锁冲突的概率最低；

## 间隙锁

## InnoDB与Myisam的最大不同点：

支持事务（Transaction）TX

采用了**行级别锁**

# 演练事务的隔离级别及现象出现

## 事务开启、提交、回滚

命令：**begin;** 开启事务或者使用 **start transaction;**

**commit;** 提交事务

**rollback;** 回滚事务

没有显式地开启一个事务，一条DML语句就是一个事务。

## 事务开启与结束的标志：

**任何一条DML语句(insert、update、delete)执行，标志事务的开启**；

- 提交：成功的结束，**将所有的DML语句操作历史记录和底层硬盘数据来一次同步；**

- 回滚：失败的结束，**将所有的DML语句操作历史记录全部清空**。

## 事务与数据库底层数据

在事物进行过程中，未结束之前，DML语句是不会更改底层数据，**只是将历史操作记录一下，在内存中完成记录**。只有在事物结束的时候，而且是成功的结束的时候，才会修改底层硬盘文件中的数据。

## 事务自动提交与手动提交

在MySQL中，默认情况下，**事务是自动提交的**，也就是说，只要执行一条DML语句就开启了事物，并且提交了事务。

关闭自动提交，需要手动开启一个事物：

begin;或者 start transaction;

DML语句;

commit;或者rollback;

## 模拟读未提交Read-Uncommitted：

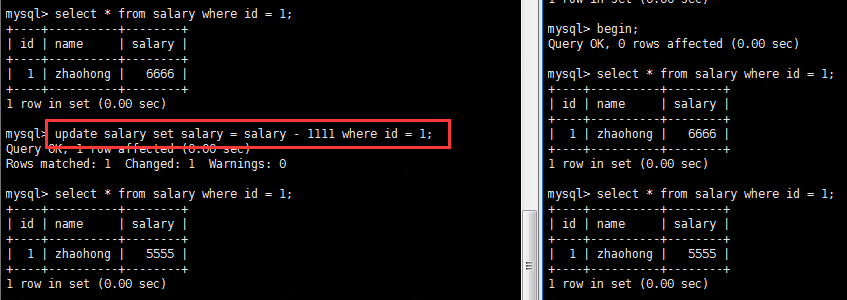
首先两端都设置事务隔离级别为Read-Uncommitted

**set tx\_isolation = 'Read-Uncommitted';**

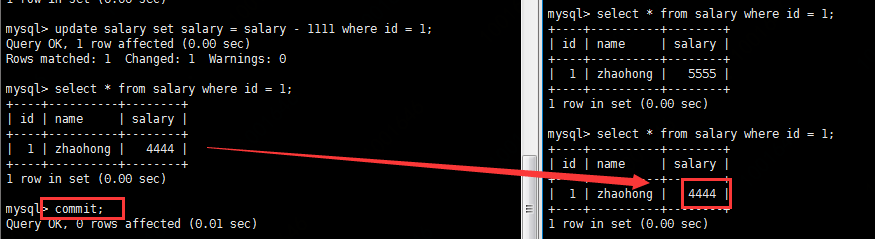
**注意：更新数据直接：**

**update salary set salary = salary - 1111 where id = 1;**

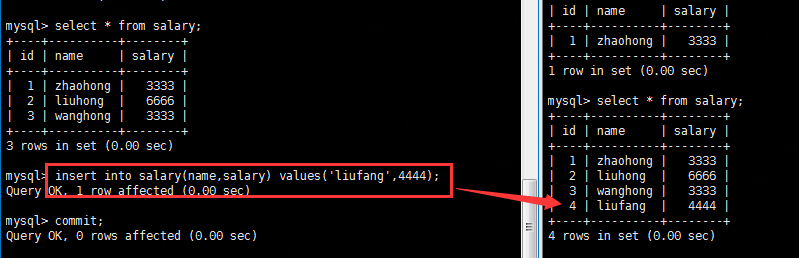
**不要先读取数据到变量中，然后处理后，再更新。**



能够读到**未提交的数据**，当然更能读到已提交的数据，出现**不可重复读现象**。

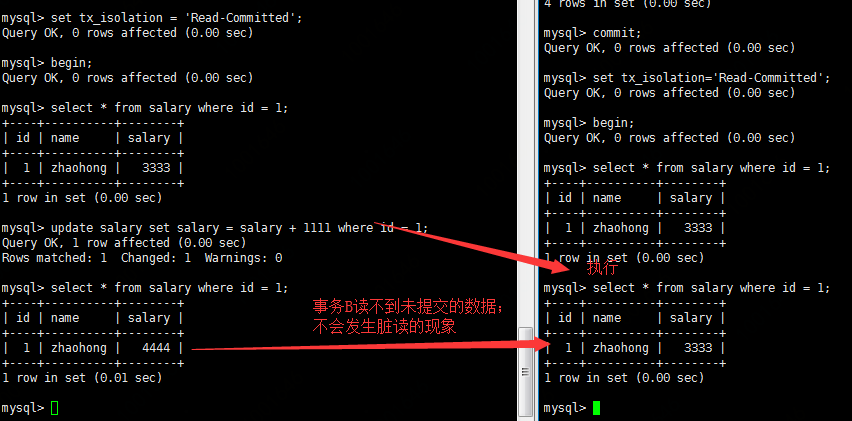


事务B读到了事务A中新增加的数据，幻读：

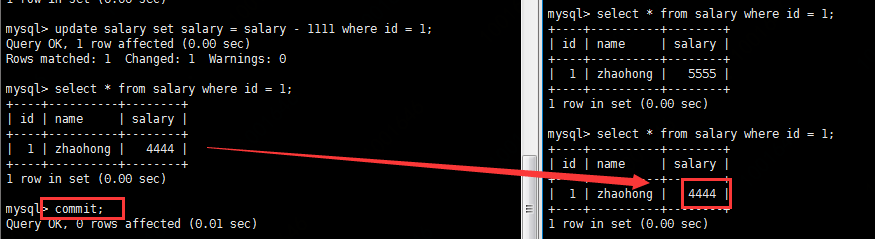


## 模拟读已提交Read-Committed

### 不会出现脏读的情况:

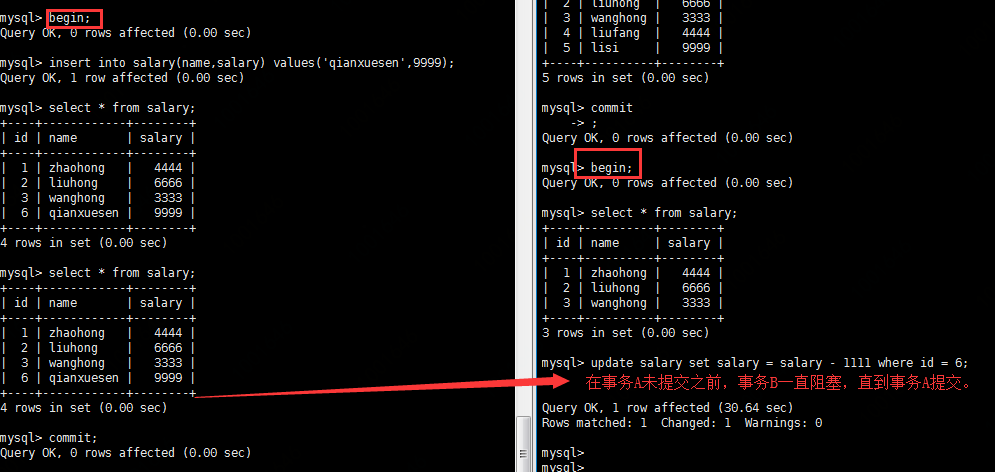


### 有可能出现不可重复读现象：



### 可能出现幻读

若事务A未提交，事务B读取不到事务A新增加的数据，若事务B试图更新事务A新增加但未提交的数据，会一直阻塞，直到事务A提交数据。

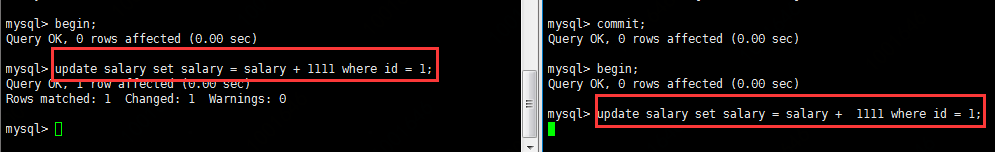


在**读已提交**隔离级别下：

阻塞：

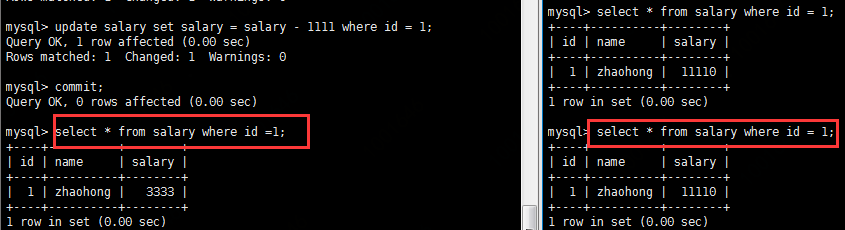
两个事务同时操作同一条记录时，先操作的成功执行，另一个会一直阻塞，直到先操作的事务提交。

幻读问题：**虽然满足可重复读问题，但是事务A增加一个新记录，事务B虽读取不到事务A新增加的记录，但是试图修改该记录时，会一直阻塞**。(相当于其实读到了，但是假装没有读到，真正操作该记录时，会一直阻塞等待)

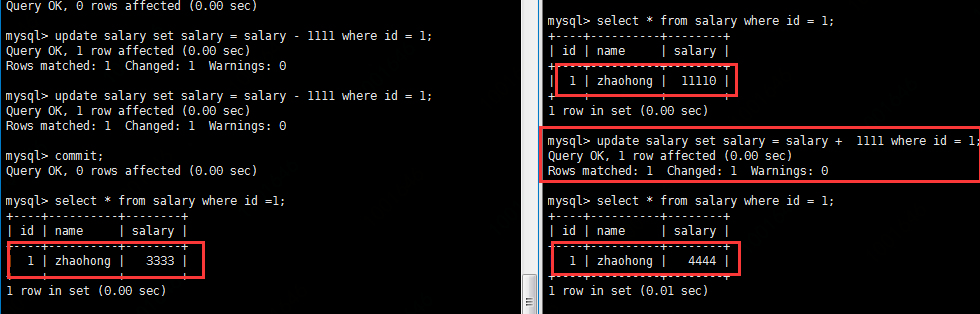


## 模拟可重复读Repeatable-Read

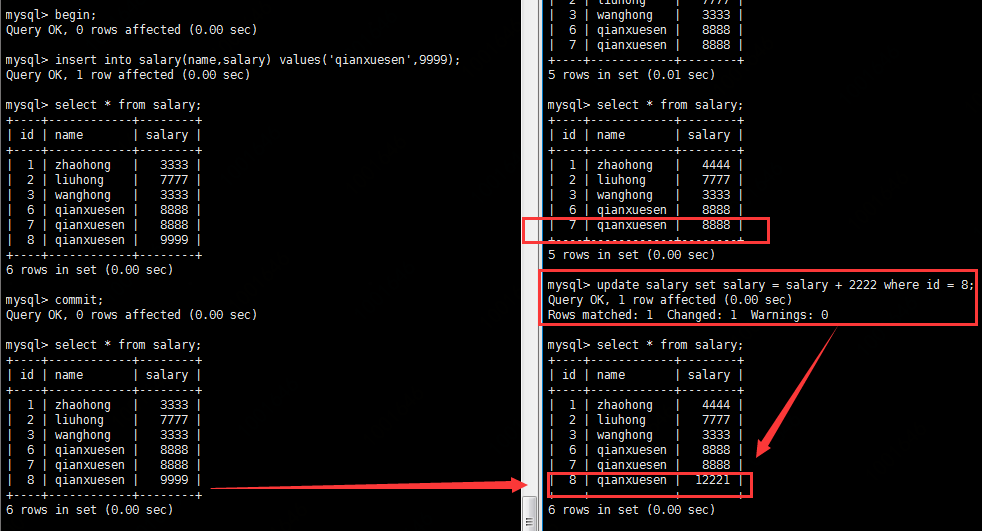
无论其他事务怎么对数据修改，原事务每一次读取的都是最初事务开始的数据。



读的数据始终就是起初读到的数据，但是，若对修改的数据修改，就会在最新的基础上修改。



幻读：



## 模拟序列化隔离级别

开启多个事务时，只能有一个事务执行DML语句，其他事务都会阻塞。不影响select语句，不会出现脏读、不可重复读、幻读现象。

# 事务隔离级别与现象的总结

## 事务隔离级别的划分依据：

行锁。

前提：多个事务对同一个表操作时，**只允许一个事务**对某一行进行**update、insert、delete**操作，第二个事务试图update、insesrt、delete时，会出现阻塞直到第一个事务提交之后，在提交之后的数据基础上改变，而允许多个事务对同一行读，其实事务的隔离级别就是根据不同事务之间的读权限进行分类:**读未提交、读已提交、可重复读、序列化**。

## 4种事务隔离级别与脏读、不可重复读、幻读的关系：

脏读-->不可重复读-->幻读级别越来越高，这三种现象的意义一定要弄十分明白。

能发生**脏读**的肯定发生不可重复读和幻读；

能发生**不可重复读**的肯定发生幻读。

### 读未提交Read-Uncommitted：

事务B可以读到事务A未提交的数据修改，读到未提交的update数据，就是脏读，读到新增的insert数据，就是幻读，这里是事务B直接select可以读到事务A新增的数据；而读已提交、可重复读都是事务B无法select到事务A新增的数据，都是事务B在update事务A新增数据时才会相当于读到。

### 读已提交Read-Committed：

事务B只能读到事务A已经提交的数据修改。不可能出现脏读现象，但是事务B在一开始时读到的数据是100，事务A修改为200，事务A提交之后，事务B读到了新的值200，因此，事务B读到了不同的值，若还有多个事务对该数据频繁修改并提交，事务B就会读到许多不同的值，这就出现了不可重复读。另外，事务A新增了数据若未提交，事务B通过select读取不到，避免了脏读，若事务A提交新增数据，事务B就可以读到，出现了幻读。另外一方面，幻读也可以体现在，事务A未提交新增数据时，事务B试图对事务A新增的数据进行update、delete时，会出现阻塞，相当于一种隐藏的读取。

### 可重复读Repeatable-Read

一个事务在开始和结束过程中与外界几乎完全隔离，除了幻读的情况。无论其他事务对数据做了怎样的修改，读取到的值都是当前事务开始读取到的值。避免了脏读和不可重复读的现象，但是仍未避免幻读的情况。其他事务新增了数据，无论是否提交，当前事务虽无法通过select查看到新增数据，但是事务修改该新增数据时，会一直阻塞，相当于一种隐藏的读取到了。

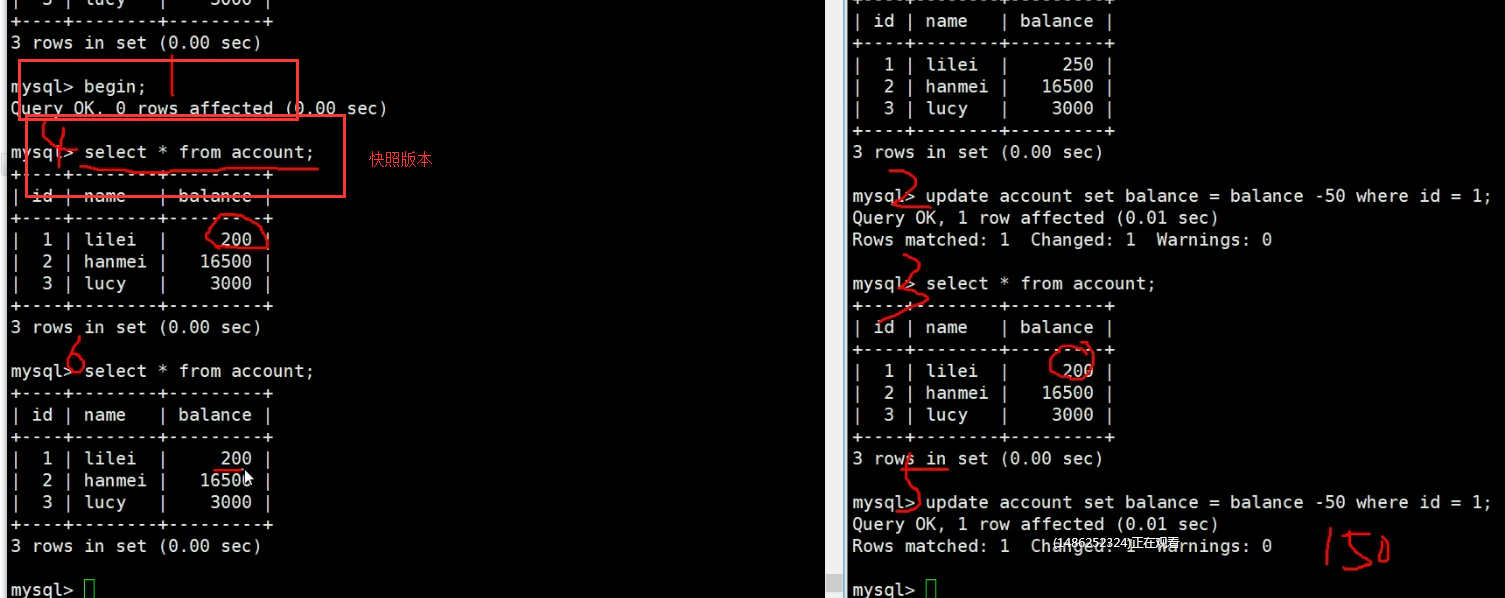
<https://blog.csdn.net/hjy930226173/article/details/79445737>

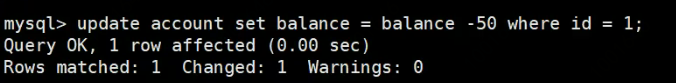
# MVCC机制

**多版本并发控制**

读的话是按照快照版本；

但是更新的话是从数据库中读取最新的值，再修改。





更新数据采用这种语句，不要采用先读取值到变量，然后再根据读到的值更新。

直接将当前值减一。