注：本次实训主要讨论事务隔离级别及其异常(19.1节)以及MVCC概念。

目录

[19.1 4种事务隔离级别和3种异常现象 1](#_Toc25134)

[19.1.1　标准的事务隔离级别 1](#_Toc20212)

[2](#_Toc6065)

[19.1.2 调整事务隔离级别 7](#_Toc25214)

[19.4 MVCC介绍 7](#_Toc23676)

## 19.1 4种事务隔离级别和3种异常现象

回顾：事务的ACID标准(而非特性)：

1. 原子性(要么全做要么不做)
2. 一致性(符合约束)
3. 隔离性(修改提交前不可见)
4. 持久性(保证数据更新后落盘)

## 19.1.1　标准的事务隔离级别

在数据库中标准的事务隔离级别有如下几种：

● 未提交读（READ-UNCOMMITTED），对应的异常现象是脏读（Dirty read）。

● 提交读（READ-COMMITTED），对应的异常现象是不可重复读（Non-repeatable read）。● 可重复读（REPEATABLE-READ），对应的异常现象是幻读（Phantom read）。

● 可序列化（SERIALIZABLE）。

注：正如书中所写，这个知识点不知为何成了面试常见考题。

**1．脏读**

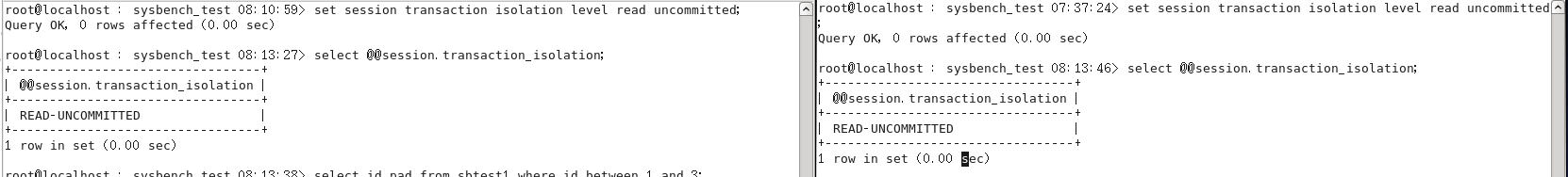
当一个事务被允许读取另外一个事务修改但**未提交的数据**时，就会发生脏读（事务隔离级别为**READ-UNCOMMITTED**，即“未提交的也是可读的”，显然事务之间互相影响没有真正隔离）。

表19-1（注：以下这些表格一定要按照时间次序，即自上而下次序执行，同一行表示可并发执行）

| **Transaction 1** | **Transaction 2** |
| --- | --- |
| begin; |  |
| select age from t1 where id =1;　return 26; | begin; |
|  | update t1 set age = 27 where id = 1;  注意这里还没有提交 |
| select age from t1 where id =1;　return 27; |  |
|  | Rollback  按理说查到的就应该是旧数据 |

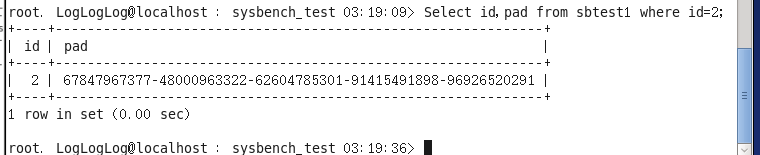
操作实例：首先设置两个会话隔离级别为未提交读

set session transaction isolation level read uncommitted;

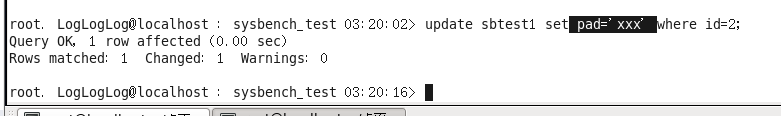


会话1开启查询事务：

Select id,pad from sbtest1 where id=2;

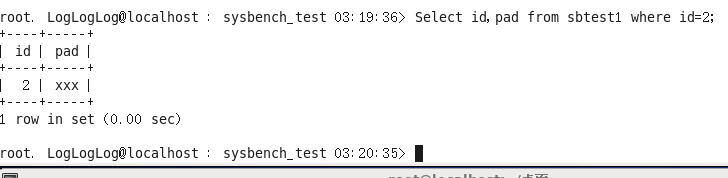


会话2开启更新事务：



update sbtest1 set pad='xxx' where id=2;

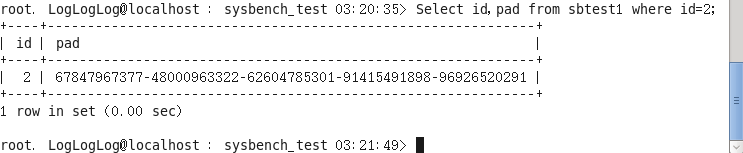
由于会话1的事务并没有与其他事务隔离，因此现在它能够读到会话2的更新数据：



然后在会话2回滚更新操作，即退回到pad=zzz：



在会话1查询到pad=zzz，正常：

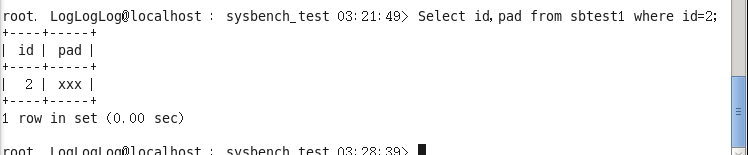


在会话2再来一次，按理说还是pad=zzz

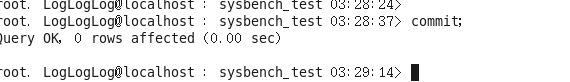
图片包含 文本

描述已自动生成

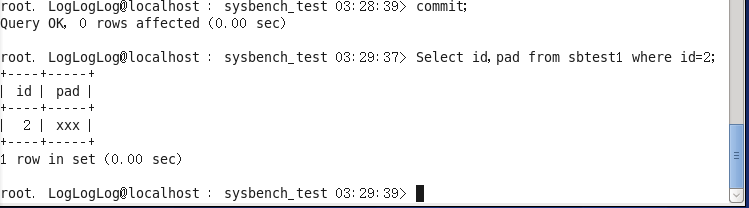
但在会话1查询，实际pad=xxx，这意味着会话1实际读到的是错误的数据(脏读)。



即使会话2提交事务：



会话1也提交事务，但查询可发现数据已经永久更改：



这时数据操作实际已经出现错误和混乱。

**2．不可重复读**

#当事务内相同的记录被检索两次，且两次得到的结果不同时，此现象称为不可重复读（事务隔离级别为READ-COMMITTED，即 “提交了的就是可以读的”，所以可能后果是读到的内容前后不一，从这个程度来说事务受到影响没有被隔离）。

表19-2

| **Transaction 1** | **Transaction 2** |
| --- | --- |
| begin; |  |
| select age from t1 where age BETWEEN 10 AND 30;  return 26; | begin; |
|  | update t1 set age = 27 where id = 1;　commit; |
| select age from t1 where id =1;  return 27;  (注：读到不一样的内容，故称“不可重复读”) |  |
| Commit |  |

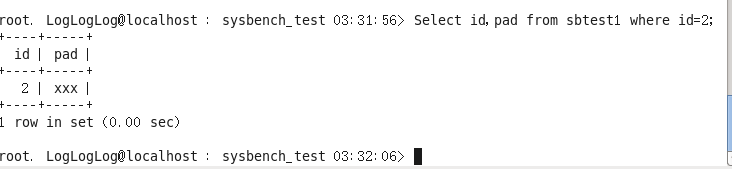
操作实例：开启两个会话，分别设置它们的事务隔离级别：

set session transaction isolation level read committed;

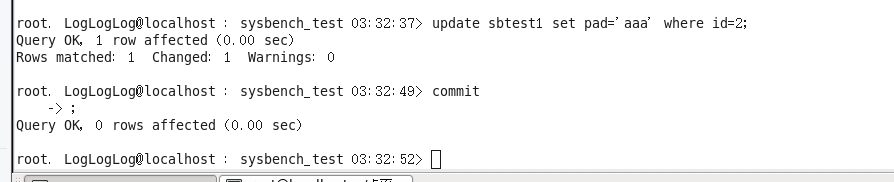
图形用户界面, 文本, 应用程序

描述已自动生成

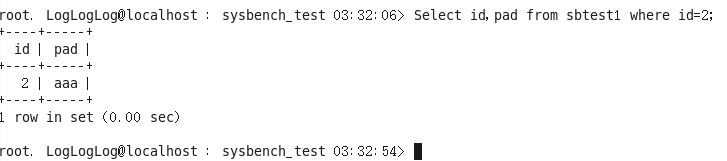
会话1开启查询事务：



然后会话2开启数据更新事务并提交：



回到会话1再次查询，数据不一样了，这就是(在一个事务内)不可重复读：



**3．幻读**

#在事务执行过程中，另一个事务**将新记录添加到正在读取的事务中**时，会发生幻读（事务隔离级别为**REPEATABLE-READ**，即“可重复读”，保证之前读到的在事务内不变）。

#当执行SELECT … WHERE语句时未对范围锁定，则可能会发生这种情况。幻读是不可重复读的一种特殊情况。

表19-3

| **Transaction 1** | **Transaction 2** |
| --- | --- |
| begin; |  |
| select age from t1 where age between 26 and 30;  　return 27; | begin; |
|  | insert into t1 (id , age) values (2,28);  　　commit;  (注：将新记录添加到正在读取的事务中) |
| select age from t1 where age between 26 and 30;  　return [27,28];(注：多出来的28叫幻行) |  |
| Commit; |  |

Set session transaction isolation level repeatable read;

操作实例：开启两个会话，分别设置它们的事务隔离级别：

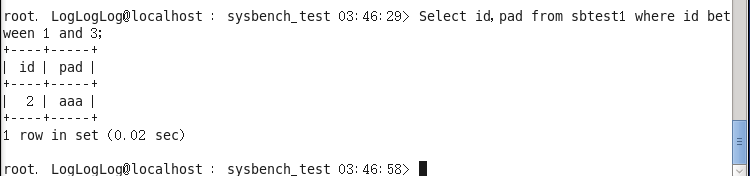
图形用户界面, 文本

描述已自动生成

文本

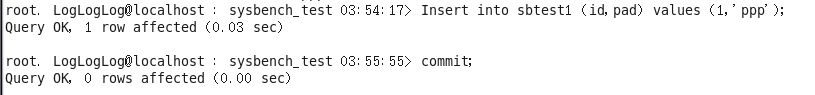
描述已自动生成

会话1开启查询事务：

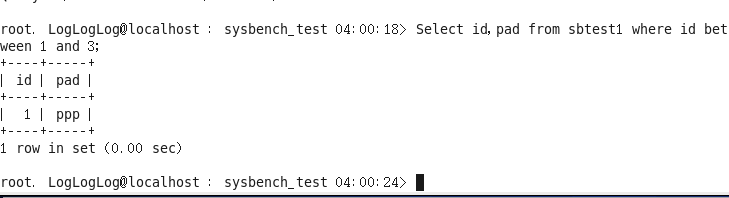


然后会话2开启数据行插入事务并提交：

Insert into sbtest1 (id,pad) values (2,'log');



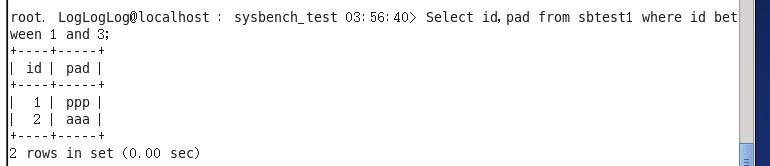
回到会话1再次查询，还是原来的数据：



并没有幻读（即返回两组不同的记录），这是因为：

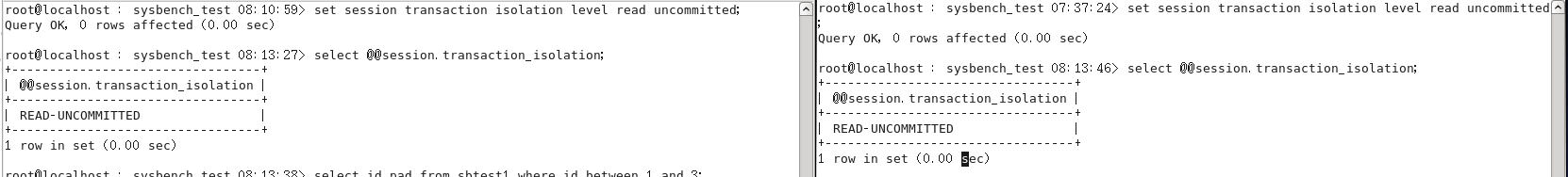
#这里需要注意，在MySQL中**增加了间隙锁**(间隙即区间，用于锁定某行之前或之后的一个区间)**防止幻读发生**，所以在MySQL中事务隔离级别为REPEATABLE-READ，不会发生以上异常现象。

也就是说，虽然事务2已经提交，但由于，但事务1并不会马上读到新增的数据行，因此并没有异常，只有在事务提交后再查询就能获得最新的数据：

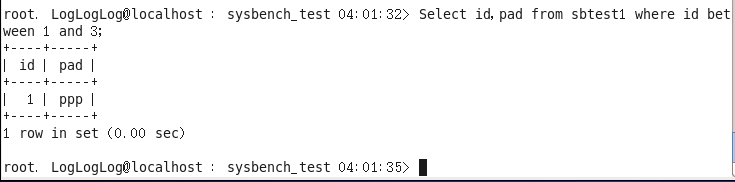


补充操作实例：其实事务隔离级别越低，会出现的问题就会越多。虽然在REPEATABLE-READ级别下实际并没有出现幻读，但在隔离级别READ-UNCOMMITTED下，不仅会出现脏读，也会出现幻读。下面恢复到未提交读级别：

set session transaction isolation level read uncommitted;

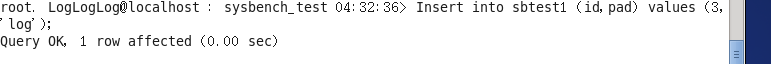


然后会话1开启查询事务(自行删掉id=1的数据行)

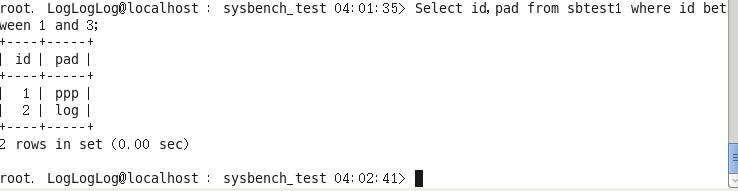


会话2开启事务后插入id=1的数据行：

Insert into sbtest1 (id,pad) values (2,'log');



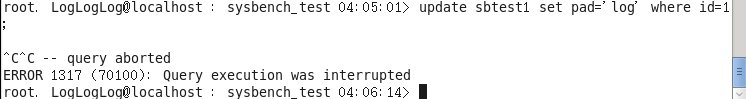
会话1自然受影响：



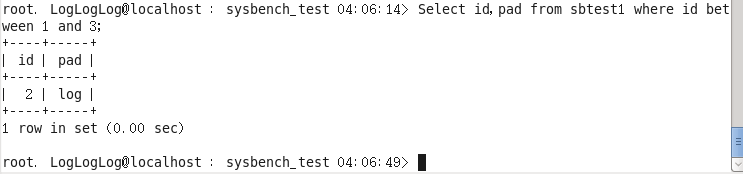
关键在于，如果会话2后来又删掉id=1的数据行，



到底这一行还有没有呢？对于会话1来说，如果这时根据上面的查询结果(14:50查过一次)，然后(在16:04之后)决定更新这行记录，实际就是在更新一条根本不存在的记录(这就是所谓的“幻行”，看到了但没有办法更新)。



当然如果这时醒悟过来，重新排查：



就会意识到要停止这种错误。由此可见，隔离级别READ-UNCOMMITTED实际很少应用。

总结(出自《高性能MySQL》,Baron Schwartz):

表格

描述已自动生成

## 19.1.2 调整事务隔离级别

在MySQL中可以通过设置transaction\_isolation参数来调整数据库事务隔离级别，默认的事务隔离级别是REPEATABLE-READ。但是为了避免发生锁等待，通常我们都将事务隔离级别设置为READ-COMMITTED。

注：实验室环境与书中一致，默认设置为READ-COMMITTED级别：



注：以上隔离级别设置均为会话级，设置全局的事务隔离变量：

SET GLOBAL transaction\_isolation = 'READ-COMMITTED'

设置好后当前会话中的事务隔离级别不会修改，新建会话就按全局设置级别隔离事务。

## 19.4 MVCC介绍

#MVCC(Multiversion concurrency control)是一种**多版本并发控制技术**，是数据库中很重要的一种技术，是DBA或者开发人员必须要了解的内容。

#未有MVCC之前：事务1正在读取一行数据，这时不能有其他事务对这行数据进行操作，因为很有可能造成事务1读取的数据不一致。

#有了MVCC之后：读写两种操作互相不阻塞。

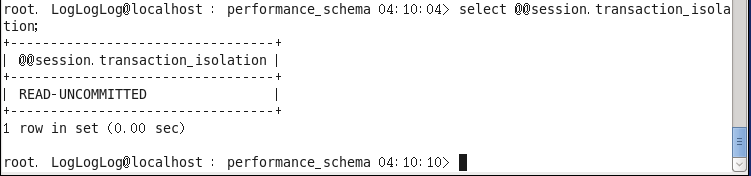
无非三种情况：读-写、读-读、写-写。读-读自然没有问题。写-写必须保证互相阻塞，否则发生脏写因此没有任何隔离级别允许两个写操作事务同时进行。唯独读-写的并发问题需要更详细讨论，它的结果取决于事务隔离级别。

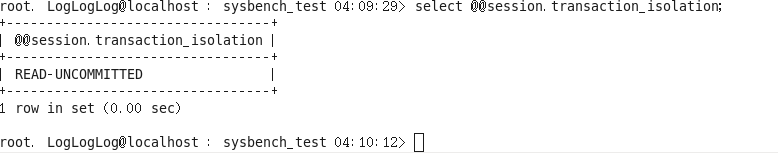
书中对此概念的解释还是过于抽象了，我们通过实例操作理解什么叫做“多版本并发控制技术”，MVCC：

| **事务1-**写 | **事务2-**读 |
| --- | --- |
| select c1 from t1; return c1 = 10 |  |
| start transaction; |  |
| update t1 set c1 = 20; |  |
|  | start transaction; |
|  | select c1 from t1; return ? |
| commit; |  |
|  | select c1 from t1; return ? |

下面对比三种事务隔离级别下读-写并发的区别：

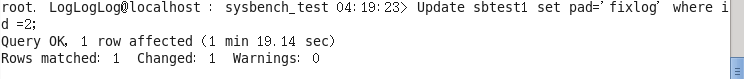
1. 在事务隔离级别为READ-UNCOMMITTED的情况下，



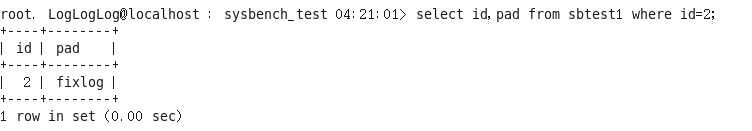


事务1先更改数据但未提交：

Update sbtest1 set pad='fixlog' where id =2;



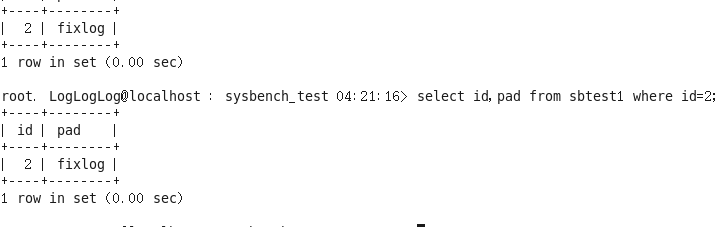
事务2开始查询，已查到未提交数据：



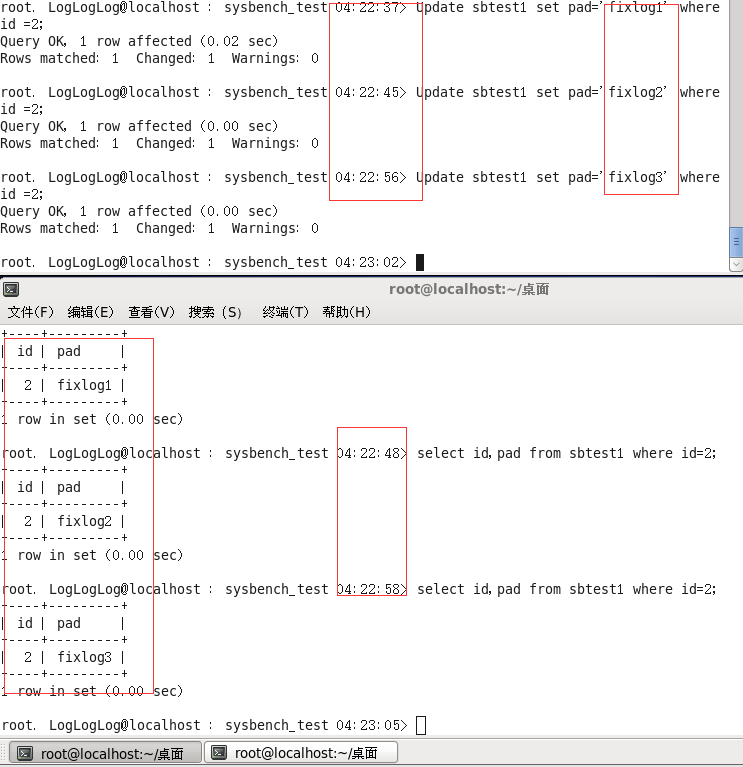
事务1提交：



事务2再次查询同一条记录，结果不变：

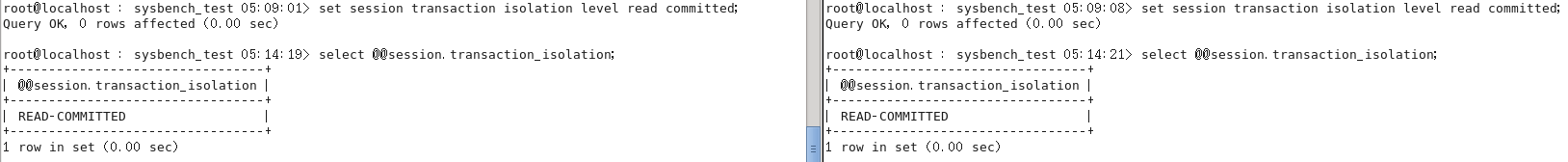


结论：事务隔离级别为READ-UNCOMMITTED下的读操作事务读到的数据始终是最新版。补充以下操作更能显示这一点，每次更新数据行之后马上查询一下：

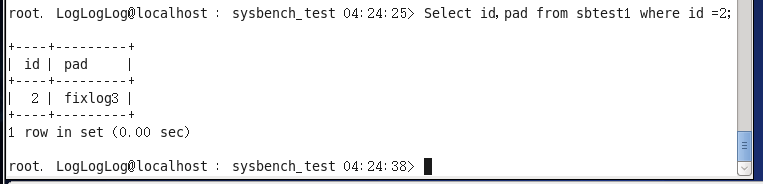


1. 在事务隔离级别为READ-COMMITTED的情况下，

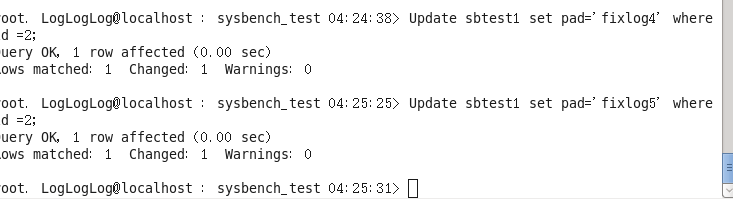
set session transaction isolation level read committed;



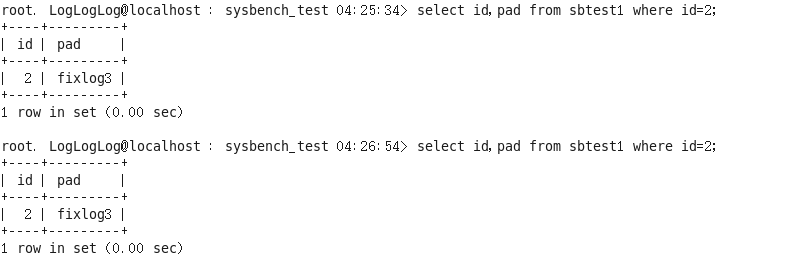
注意事务1未开始时读数据的结果：



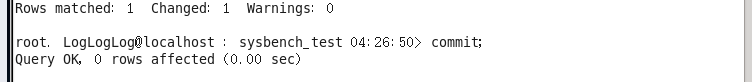
然后事务1多次更改数据，但未提交：

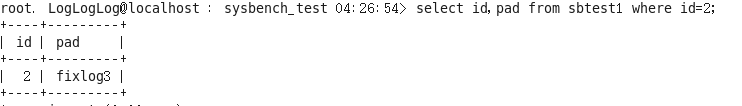


无论事务1怎么更改，事务2那边读到的都是原来事务1未开始的数据，以下两次读取数据分别在以上事务1更改数据行之后读取：



事务1提交更改之后，事务2能读到最新版的数据：

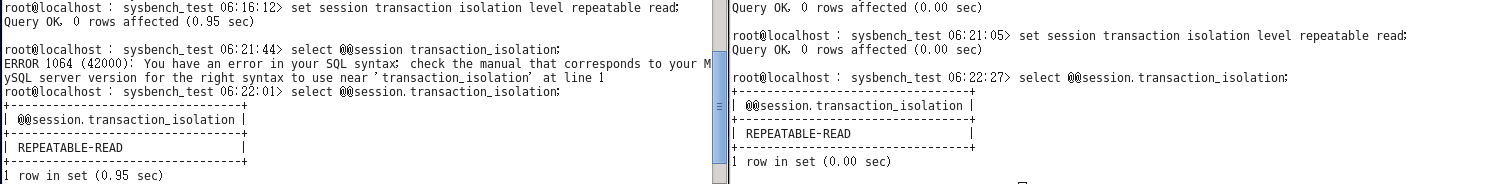




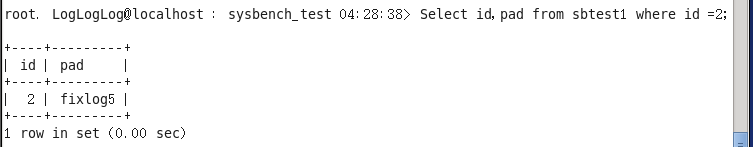
结论：事务隔离级别为READ-COMMITTED下，在写操作事务未提交之前，读操作事务读到的数据是最旧版本的数据；写操作事务提交之后，读操作事务能读到其写入的最新版数据。

3. 在事务隔离级别为REPEATABLE-READ的情况下，

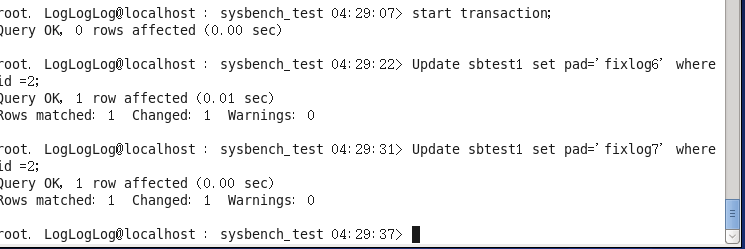
set session transaction isolation level REPEATABLE READ;



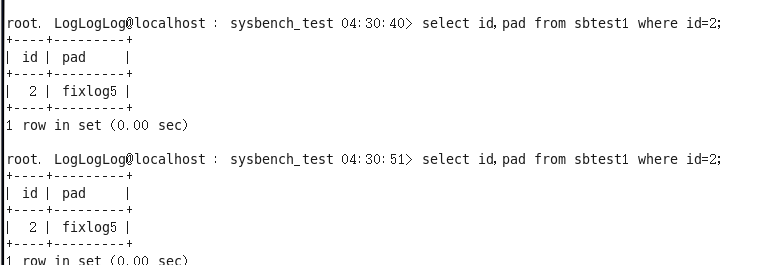
注意事务1未开始时读数据的结果：



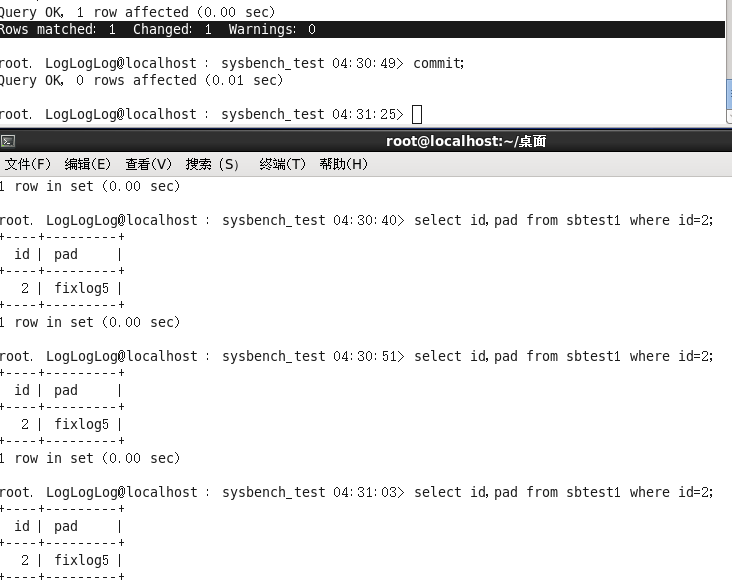
然后事务1多次更改数据，但未提交：



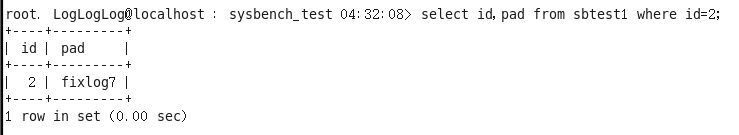
事务2对应地先后两次查询，内容还是事务1未开始前的旧版本：



事务1提交之后(对比READ-COMMITTED级别)，事务2查到的仍然是旧版数据：



我们只有事务2提交之后才查到最新版数据：



结论：读操作事务整个过程只会读到一个版本的数据，就是事务开始时那个版本的数据。

总结(部分参考了《Mysql是怎样运行的》，21.3节)：

1. MVCC“多版本并发控制技术”中的“版本”相当于“快照”，由于事务之间并发执行，所以实际上会有若干版本同时存在于一个系统中。Mysql根据隔离策略控制事务可以读到哪个版本的数据。

2. 在写操作事务开始前系统会把原记录放入undo表空间，所以READ-COMMITTED和REPEATABLE-READ级别下读操作事务总能读到原来写操作事务未开始时的数据。

3. “版本”在技术上以一种称为ReadView的东西实现。READ-COMMITTED级别在每次读取数据前都生成一个ReadView，而REPEATABLE-READ级别只在第一次读取数据时生成一个ReadView，这导致READ-COMMITTED级别下读操作事务能及时在写操作事务提交后获得新版数据，而REPEATABLE-READ级别能保证整个读操作事务内数据保持不变。