Mysql事务相关测试脚本

1、查看 mysql 隔离级别

查看当前会话的隔离级别

```
mysql> select @@tx_isolation;
+-----------------+
| @@tx_isolation  |
+-----------------+
| REPEATABLE-READ |
+-----------------+
1 row in set (0.00sec)
```

查看全局的隔离级别

```
mysql> select @@global.tx_isolation;
+-----------------------+
| @@global.tx_isolation |
+-----------------------+
| REPEATABLE-READ       |
+-----------------------+
1 row in set (0.00sec)
```

2、设置 mysql 隔离级别

设置当前会话的隔离级别

```
mysql> set session transaction isolation level repeatable read;
Query OK, 0 rows affected (0.01sec)
```

设置全局的隔离界别

```
mysql> set global transaction isolation level repeatable read;
Query OK, 0 rows affected (0.00sec)
```

3、例子：我们有一个 transaction 的数据库，里面有一张 product 商品数据表，然后打开两个终端 A B 来测试这四种隔离级别。

切换到transaction库，并查看product表信息

```
mysql> use transaction;
Database changed

mysql> select * from product;
+----+--------+--------+-------+-------------+
| id | name   | amount | state | create_time |
+----+--------+--------+-------+-------------+
|  1 | 苹果   |     20 |     1 |           0 |
+----+--------+--------+-------+-------------+
1 row in set (0.00sec)
```

未提交读 read uncommitted 演示

终端 A 设置当前会话隔离级别为 read uncommitted

```
mysql> set session transaction isolation level read uncommitted;
Query OK, 0 rows affected (0.00sec)
```

终端 B 设置当前会话隔离级别为 read uncommitted

```
mysql> set session transaction isolation level read uncommitted;
Query OK, 0 rows affected (0.00sec)
```

查询终端 A 开启事务

```
mysql> begin;
Query OK, 0 rows affected (0.00sec)
```

查询 product 表苹果价钱 = 20

```
mysql> select * from product;
+----+--------+--------+-------+-------------+
| id | name   | amount | state | create_time |
+----+--------+--------+-------+-------------+
|  1 | 苹果   |     20 |     1 |           0 |
```

```
+----+--------+--------+-------+-------------+
1 row in set (0.00sec)
```

终端 B 开启事务

```
mysql> begin;
Query OK, 0 rows affected (0.00sec)
```

修改苹果价钱 20 - 5 = 15

```
mysql> update product set amount = amount - 5 where id = 1;
Query OK, 1 row affected (0.00sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from product;
+----+--------+--------+-------+-------------+
| id | name   | amount | state | create_time |
+----+--------+--------+-------+-------------+
|  1 | 苹果   |     15 |     1 |           0 |
+----+--------+--------+-------+-------------+
1 row in set (0.00sec)
```

再次查看终端 A 苹果价钱 = 20，终端 B 没有将事务提交，但是终端 A 可以读取到终端 B DML操作之后的结果

如果终端 B 将事务回滚，那么终端 A 读取到的数据就是脏数据

```
mysql> select * from product;
+----+--------+--------+-------+-------------+
| id | name   | amount | state | create_time |
+----+--------+--------+-------+-------------+
|  1 | 苹果   |     15 |     1 |           0 |
+----+--------+--------+-------+-------------+
1 row in set (0.00sec)
```

终端 B 事务回滚

```
mysql> rollback;
Query OK, 0 rows affected (0.00sec)
```

苹果价钱没有修改

```
mysql> select * from product;
+----+--------+--------+-------+-------------+
| id | name   | amount | state | create_time |
```

```
+----+--------+--------+-------+-------------+
|  1 | 苹果   |     20 |     1 |           0 |
+----+--------+--------+-------+-------------+
1 row in set (0.00sec)
```

再次查看终端 A 苹果价钱还是20，那么刚才读取到的 15 就是脏数据

```
mysql> select * from product;
+----+--------+--------+-------+-------------+
| id | name   | amount | state | create_time |
+----+--------+--------+-------+-------------+
|  1 | 苹果   |     20 |     1 |           0 |
+----+--------+--------+-------+-------------+
1 row in set (0.00sec)
```

在 read uncommitted 级别，事物中的修改，即使没有提交，对其他事务也都是可见的。事务可以读取未提交的数据，这也被称为脏读 (Dirty Read)。这个级别会导致很多问题，从性能上说 read uncommitted 不会比其他的级别好太多，但是缺乏其他级别的很多好处，除非真的有非常必要的理由，在实际应用中有一般很少用到。

读已提交 read committed 演示

终端 A 设置当前会话隔离级别为 read committed

```
mysql> set session transaction isolation level read committed;
Query OK, 0 rows affected (0.00sec)
```

终端 B 设置当前会话隔离级别为 read committed

```
mysql> set session transaction isolation level read committed;
Query OK, 0 rows affected (0.00sec)
```

终端 A 开启事务查询苹果价格

```
mysql> begin;
Query OK, 0 rows affected (0.00sec)

mysql> select * from product;
+----+--------+--------+-------+-------------+
| id | name   | amount | state | create_time |
+----+--------+--------+-------+-------------+
|  1 | 苹果   |     20 |     1 |           0 |
+----+--------+--------+-------+-------------+
1 row in set (0.00sec)
```

终端 B 开启事务

```
mysql> begin;
Query OK, 0 rows affected (0.00sec)
```

苹果价格 20 - 5 = 15

```
mysql> update product set amount = amount - 5 where id = 1;
Query OK, 1 row affected (0.00sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from product;
+----+--------+--------+-------+-------------+
| id | name   | amount | state | create_time |
+----+--------+--------+-------+-------------+
|  1 | 苹果   |     15 |     1 |           0 |
+----+--------+--------+-------+-------------+
1 row in set (0.00sec)
```

这时候 终端 B 的事务还没有提交，再次查询终端 A 发现数据并没有变，解决了刚才脏读的问题

```
mysql> select * from product;
+----+--------+--------+-------+-------------+
| id | name   | amount | state | create_time |
+----+--------+--------+-------+-------------+
|  1 | 苹果   |     20 |     1 |           0 |
+----+--------+--------+-------+-------------+
1 row in set (0.00sec)
```

终端 B 提交事务，事务执行成功

```
mysql> commit;
Query OK, 0 rows affected (0.01sec)
```

苹果价格成功改为 15 元，此时已经成功改写数据磁盘中的结果

```
mysql> select * from product;
+----+--------+--------+-------+-------------+
| id | name   | amount | state | create_time |
+----+--------+--------+-------+-------------+
|  1 | 苹果   |     15 |     1 |           0 |
+----+--------+--------+-------+-------------+
1 row in set (0.00sec)
```

终端 A 查看苹果价钱变成了 15 元，在一个事务内两次读到的数据产生变化，因此称为不可重复读

```
mysql> select * from product;
+----+--------+--------+-------+-------------+
| id | name   | amount | state | create_time |
+----+--------+--------+-------+-------------+
|  1 | 苹果   |     15 |     1 |           0 |
+----+--------+--------+-------+-------------+
1 row in set (0.00sec)
```

read committed 这种级别很好的解决了脏读的问题，但是又产生了不可重复读的问题。一种更易理解的说法是：在一个事务内，多次读同一个数据。在这个事务还没有结束时，另一个事务也访问该同一数据并修改数据。那么，在第一个事务的两次读数据之间。由于另一个事务的修改，那么第一个事务两次读到的数据可能不一样，这样就发生了在一个事务内两次读到的数据是不一样的，因此称为不可重复读，即原始读取不可重复。多数的数据库系统默认的隔离级别都是 read committed。

可重复读 repeatable read 演示

终端 A 设置当前会话隔离级别为 repeatable read

```
mysql> set session transaction isolation level repeatable read;
Query OK, 0 rows affected (0.00sec)
```

开启事务

```
mysql> begin;
Query OK, 0 rows affected (0.00sec)
```

苹果价钱 = 20 元

```
mysql> select * from product;
+----+--------+--------+-------+-------------+
| id | name   | amount | state | create_time |
+----+--------+--------+-------+-------------+
|  1 | 苹果   |     20 |     1 |           0 |
+----+--------+--------+-------+-------------+
1 row in set (0.00sec)
```

终端 B 设置当前会话隔离级别为 repeatable read

```
mysql> set session transaction isolation level repeatable read;
Query OK, 0 rows affected (0.00sec)
```

开启事务

```
mysql> begin;
Query OK, 0 rows affected (0.00sec)
```

苹果价钱 = 20 元

```
mysql> select * from product;
+----+--------+--------+-------+-------------+
| id | name   | amount | state | create_time |
+----+--------+--------+-------+-------------+
|  1 | 苹果   |     20 |     1 |           0 |
+----+--------+--------+-------+-------------+
```

价钱减去 5 元

```
mysql> update product set amount = amount - 5 where id = 1;
Query OK, 1 row affected (0.00sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from product;
+----+--------+--------+-------+-------------+
| id | name   | amount | state | create_time |
+----+--------+--------+-------+-------------+
|  1 | 苹果   |     15 |     1 |           0 |
+----+--------+--------+-------+-------------+
1 row in set (0.00sec)
```

关闭事务

```
mysql> commit;
Query OK, 0 rows affected (0.00sec)
```

终端 A 中的苹果价钱 = 20 元，与第一次查询结果一致，没有出现不可重复读问题

```
mysql> select * from product;
+----+--------+--------+-------+-------------+
| id | name   | amount | state | create_time |
+----+--------+--------+-------+-------------+
|  1 | 苹果   |     20 |     1 |           0 |
+----+--------+--------+-------+-------------+
1 row in set (0.00sec)
```

将价苹果价钱减去 5 元

```
mysql> update product set amount = amount - 5 where id = 1;
Query OK, 1 row affected (0.00sec)
```

Rows matched: 1 Changed: 1 Warnings: 0

价钱没有从 20 - 5 = 15，反而成了 10 元，这是因为终端 A 中操作的值是从终端 B 中最后的 15 - 5 = 10 算出来的。
但是数据的一致性倒是没有被破坏。
可重复读的隔离级别下使用了 MVCC 机制，select 操作不会更新版本号，是快照读(历史版本)，insert、update、delete 会更新版本号，是当前读(当前版本)。

```
mysql> select * from product;
+----+--------+--------+-------+-------------+
| id | name   | amount | state | create_time |
+----+--------+--------+-------+-------------+
|  1 | 苹果   |     10 |     1 |           0 |
+----+--------+--------+-------+-------------+
1 row in set (0.00sec)
```

重新打开终端 B 开启事务

```
mysql> begin;
Query OK, 0 rows affected (0.00sec)
```

插入一条香蕉

```
mysql> insert into product (`name`, `amount`) values ('香蕉', '30');
Query OK, 1 row affected, 1 warning (0.00sec)

mysql> select * from product;
+----+--------+--------+-------+-------------+
| id | name   | amount | state | create_time |
+----+--------+--------+-------+-------------+
|  1 | 苹果   |     15 |     1 |           0 |
|  2 | 香蕉   |     30 |     1 |           0 |
+----+--------+--------+-------+-------------+
2 rows in set (0.00sec)
```

关闭事务

```
mysql> commit;
Query OK, 0 rows affected (0.00sec)
```

终端 A 中的还是只有一条苹果数据，没有出现幻读

```
mysql> select * from product;
+----+--------+--------+-------+-------------+
| id | name   | amount | state | create_time |
+----+--------+--------+-------+-------------+
|  1 | 苹果   |     10 |     1 |           0 |
+----+--------+--------+-------+-------------+
1 row in set (0.00sec)
```

所谓幻读，指的是当某个事务在读取某个范围内的记录时，另外一个事务又在该范围内插入了新的记录，当之前的事务再次读取该范围的记录时，会产生幻行(Phantom Row)。

串行化 serializable 演示

终端 A 设置当前会话隔离级别为 serializable

```
mysql> set session transaction isolation level serializable;
Query OK, 0 rows affected (0.00sec)
```

开启事务

```
mysql> begin;
Query OK, 0 rows affected (0.00sec)
```

终端 B 设置当前会话隔离级别为 serializable

```
mysql> set session transaction isolation level serializable;
Query OK, 0 rows affected (0.00sec)
```

开启事务

```
mysql> begin;
Query OK, 0 rows affected (0.00sec)
```

终端 A

查询 amount = 10 的商品

```
mysql> select * from product where amount = 10;
+----+--------+--------+-------+-------------+
| id | name   | amount | state | create_time |
+----+--------+--------+-------+-------------+
|  1 | 苹果   |     10 |     1 |           0 |
```

```
+----+--------+--------+-------+-------------+
1 rows in set (0.00sec)
```

将 amount = 10 的 name='苹果' 更新 name='香蕉'

```
mysql> update product set name = '香蕉' where author = 10;
Query OK, 1 row affected (0.00sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

终端 B

查询 amount = 10 的商品，此时无法查询成功（因为事务 A 对 author 增加了排它锁，所以事务 B 无法再次增加共享锁），必须等事务 A commit 之后才可以成功

```
mysql> select * from my_innodb where author = 10;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

这是最高的隔离级别，避免了脏读、幻读、实现了可重复读。serializable 会对操作的数据都加锁，读加共享锁，写加排他锁，读写互斥。所以可能导致大量的超时和锁争用的问题，实际应用中很少用这个隔离级别，只有在非常需要确保数据一致性而且可以接受没有并发的情况下，才会考虑该级别。