* **MySQL事务隔离级别详解**

SQL标准定义了4类隔离级别，包括了一些具体规则，用来限定事务内外的哪些改变是可见的，哪些是不可见的。低级别的隔离级一般支持更高的并发处理，并拥有更低的系统开销。

* Read Uncommitted（读取未提交内容）

在该隔离级别，所有事务都可以看到其他未提交事务的执行结果。本隔离级别很少用于实际应用，因为它的性能也不比其他级别好多少。读取未提交的数据，也被称之为脏读（Dirty Read）。

* Read Committed（读取提交内容）

这是大多数数据库系统的默认隔离级别（但不是MySQL默认的）。它满足了隔离的简单定义：一个事务只能看见已经提交事务所做的改变。这种隔离级别也支持所谓的不可重复读（Nonrepeatable Read），因为同一事务的其他实例在该实例处理其间可能会有新的commit，所以同一select可能返回不同结果。

* Repeatable Read（可重读可重读可重读可重读）

这是MySQL的默认事务隔离级别，它确保同一事务的多个实例在并发读取数据时，会看到同样的数据行。不过理论上，这会导致另一个棘手的问题：幻读（Phantom Read）。简单的说，幻读指当用户读取某一范围的数据行时，另一个事务又在该范围内插入了新行，当用户再读取该范围的数据行时，会发现有新的“幻影” 行。InnoDB和Falcon存储引擎通过多版本并发控制（MVCC，Multiversion Concurrency Control）机制解决了该问题。

* Serializable（可串行化）

这是最高的隔离级别，它通过强制事务排序，使之不可能相互冲突，从而解决幻读问题。简言之，它是在每个读的数据行上加上共享锁。在这个级别，可能导致大量的超时现象和锁竞争。

* **术语解释**

这四种隔离级别采取不同的锁类型来实现，若读取的是同一个数据的话，就容易发生问题。

* 脏读(Drity Read)

某个事务已更新一份数据，另一个事务在此时读取了同一份数据，由于某些原因，前一个RollBack了操作，则后一个事务所读取的数据就会是不正确的。

* 不可重复读(Non-repeatable read)

在一个事务的两次查询之中数据不一致，这可能是两次查询过程中间插入了一个事务更新的原有的数据。

* 幻读(Phantom Read)

在一个事务的两次查询中数据笔数不一致，例如有一个事务查询了几列(Row)数据，而另一个事务却在此时插入了新的几列数据，先前的事务在接下来的查询中，就会发现有几列数据是它先前所没有的。

* MySql图解

在MySQL中，实现了这四种隔离级别，分别有可能产生问题如下所示：

|  |  |  |  |
| --- | --- | --- | --- |
| **隔离级别** | **脏读** | **不可重复读** | **幻读** |
| 读未提交(Read Uncommitted) | **✓** | **✓** | **✓** |
| 读已提交(Read Committed) | **🗶** | **✓** | **✓** |
| 可重复读(Repeatable Read) | **🗶** | **🗶** | **✓** |
| 可串行化(Serializable) | **🗶** | **🗶** | **🗶** |

* **演示**
* 准备工作
* 创建数据库

CREATE DATABASE itcast CHARACTER SET UTF8;

* 创建表

create table users(

id varchar(32),

name varchar(50)

);

* 添加数据

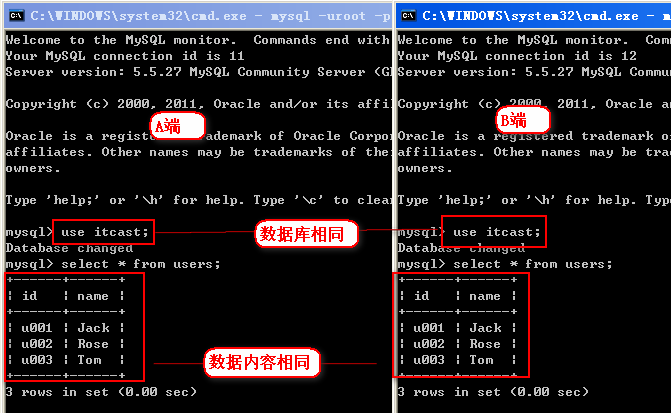
insert into users(id,name) values('u001','Jack');

insert into users(id,name) values('u002','Rose');

insert into users(id,name) values('u003','Tom');

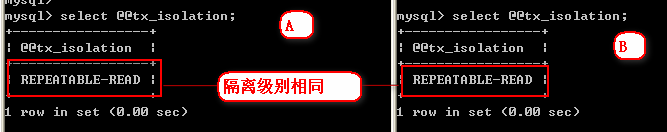
* 打开两个命令窗口

进入相同数据库，并检查当前表内容为相同数据如下



* 查看隔离级别

在A、B两端执行select @@tx\_isolation;

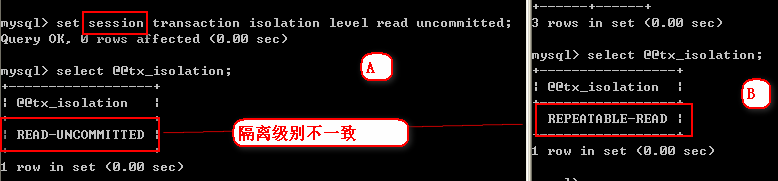


* Read Uncommitted（读取未提交内容）

1、修改A端的隔离级别为read uncommitted – 读未提交

set session transaction isolation level read uncommitted;

2、然后再查看当前隔离级别

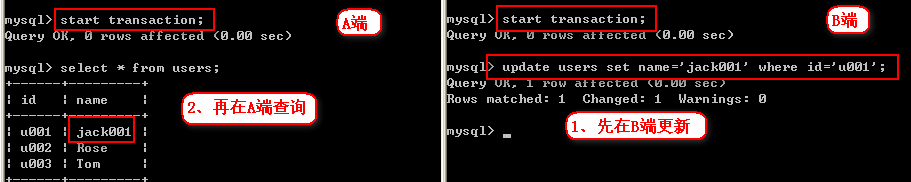


3、在A、B两端都开启事务

start transaction;

4、在B端修改数据，update users set name='jack001' where id='u001';

在A端查询数据，select \* from users;



5、此时B端执行回滚操作: Rollback;

再在A端进行查询

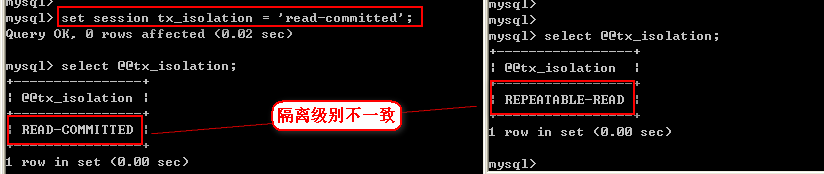


对于A端来说，读取了B端还没有提交的数据。

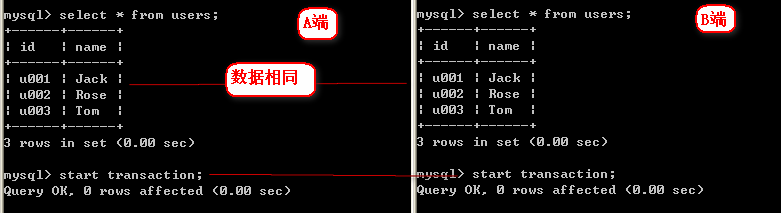
* Read Committed（读取提交内容）

1、修改A端的隔离级别为read committed

set session tx\_isolation = 'read-committed';



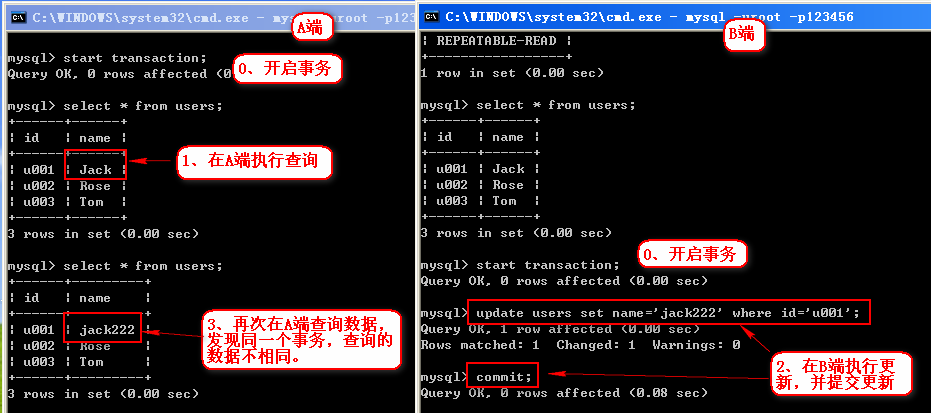
2、确定数据相同，并同时开启事务



3、在A端进行查询：select \* from users;

在B端修改一行记录并提交：update users set name='jack222' where id='u001';

再回到A端进行查询，发现在同一个事务内，两次查询的结果不一样

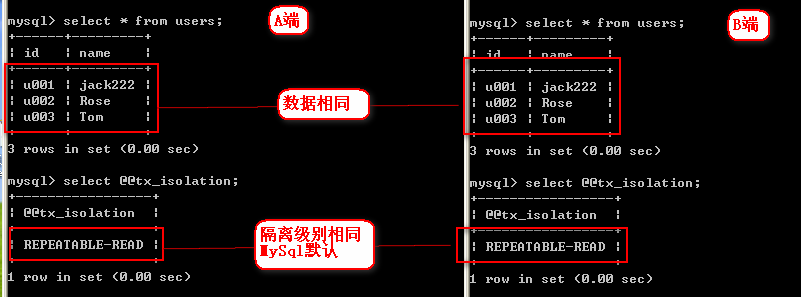


* Repeatable Read（可重读可重读可重读可重读）

1、设置A端隔离级别Repeatable Read

set session tx\_isolation = 'repeatable-read';

2、查看隔离级别以及数据



3、同时开启事务

在A端进行查询：select \* from users;

然后在B端修改内容并提交事务：update users set name = 'jack333' where id='u001';

最后再在A端的同一事务内进行查询，发现结果一致

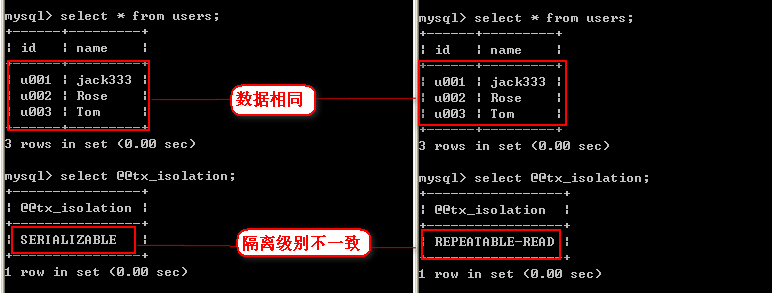


* Serializable（可串行化）

1、设置A端隔离级别Serializable

set session transaction isolation level serializable;

2、确定数据和事务级别



3、同时开启事务

在A端进行查询：select \* from users;

然后在B端修改内容并提交事务：update users set name = 'jack444' where id='u001';

此时发现B的代码并没有执行，因为它在等A提交之后它才执行。类似于线程同步的概念

