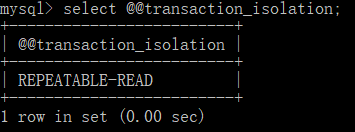
### MySQL事务隔离级别和传播行为

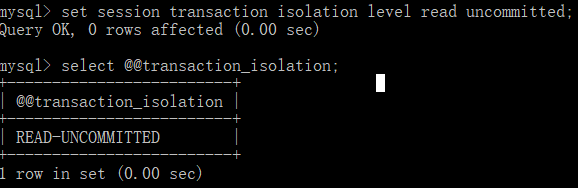
#### 隔离级别：

查询mysql默认的隔离级别



#### Read Uncommitted（本事务可以读取到其它事务未提交数据）

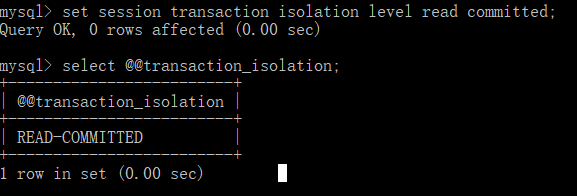
将B事务的隔离级别设置为”Read Uncommitted”



A事务开启，插入一条数据后，B事务查询后看到了新插入的数据。

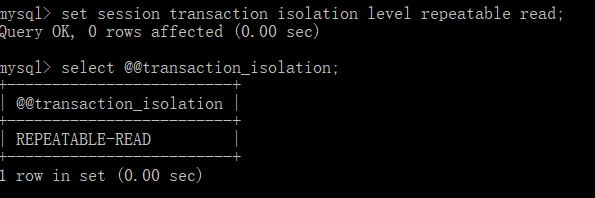
之后A事务rollback，B事务再也看不到该条数据，仿佛那条数据没有存在过一样 - 脏读。

#### Read Committed（本事务只能读取其他事务已提交数据）



A事务如果进行了update操作，但是没有提交，B事务读取的还是A未提交的数据，但是一旦A事务提交了，B事务就能够读取A提交的最新的数据。想一想，B在一个事务里面读取的数据不一致 - 不可重复读。

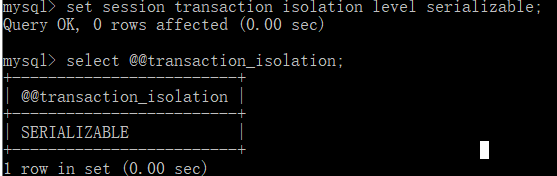
#### Repeatable Read(可重复读 - 本事务总是读取一致的数据，相当于在事务开启前对数据进行快照)



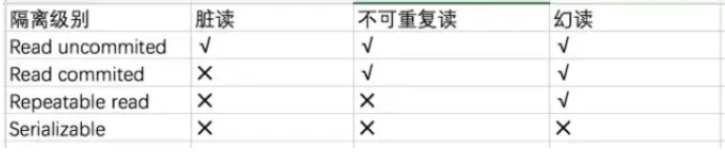
有种情况，A事务对数据条数进行更改，例如我们演示的，A事务insert了一条数据，B事务中不能查询到，A事务提交后B事务也不能查询到，直到B事务提交才能看到新添加的数据。

这种影响数据条数的情况，B事务前后读取不一致的现象 - 幻读。

#### Serializable(可串行读 - 事务排队)



B事务会等待其他的事物都完成提交后，才执行。 在等待的过程中，可能事物会超时。



#### 传播行为：

Propagation.REQUIRED 需要事务，如果当前存在事务就沿用，否则新建一个新的事务

Propagation.REQUIRES\_NEW 无论当前事务是否存在，都会创建新事物

Propagation.NESTED 在当前方法调用子方法时，如果子方法发生异常只回滚子方法执行过的SQL，而不回滚当前事务，所以就是可以“一部分成功”，有一个前提是外层要使用try-catch

父事务：

|  |
| --- |
| Transactional(isolation = Isolation.REPEATABLE\_READ, propagation = Propagation.REQUIRED)  @Override  public Integer insertBatch(List<Yesterday> yesList) {  int num = 0;  for(int i = 0; i < yesList.size(); i++) {  Yesterday yesterday = yesList.get(i);  try {  num += yesterdaySerivce.insert(yesterday);  } catch (Exception e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }    }  return num;  } |

子事务

|  |
| --- |
| @Transactional(isolation = Isolation.REPEATABLE\_READ, propagation = Propagation.NESTED)  @Override  public Integer insert(Yesterday yesterday) {  if(yesterday.getCid() == 4) {  throw new RuntimeException("Exeception Drill");  }  return yesterdayMapper.insert(yesterday);  } |